SINCLAIR

Every month £1.75 March 1990

4 to 7 9 11 INVERTED QL
14 17 18 20
26 30 36 46

QL WORLD

MULTIBASIC

Now the QL can keep more in the air than ever before

SOFTWARE FILE:
Musiql
Here We Go
SBMon

SUPERBASIC
The QL character set

USING FLASHBACK 2
And converting from Flashback 1

PROGRAM OF THE MONTH
On the trail with Plant Life

ISSN 0951-9335

03

9 770951 933009

SINCLAIR QL WORLD

Editor
Helen Armstrong

Chief Sub Editor
Harold Mayes MBE

Production Manager
Nick Fry

Designer
Chris Winch

Advertising Sales
Jason Newman

Magazine Services
Sheila Baker

Advertising Production
Michelle Evans

Group Advertising Manager
Claire Baxter

Group Editor
Carlo Jolly

Publisher
Perry Trevers

Managing Director
Peter Welham

Financial Director
Brendan McGrath

Chief Executive
Richard Hease

Microdrive Exchange
089 283 4783/2952
(2 lines) TIL

Sinclair QL World
Greencoat House
Francis Street
London SW1 1DG
Telephone 01-834 1717
Fax 01-828 0270
Telex 9419564 FOCUS G
ISSN 026806X

Unfortunately, we are no longer able to answer enquiries made by telephone. If you have any comments or difficulties, please write to The Editor, Open Channel, Trouble Shooter, or Psion Solutions. We will do our best to deal wity your problem in the magazine, thuough we cannot guarantee individual replies.
Back issues are available from the publisher price £2 U.K., £2.75 Europe. Overseas rates on request. Please telephone 089 283 4783 to check availability.
Published by Focus Magazine Ltd., London. S M Distribution, Streatham, London SW1. 01 677 8111. Subscription information from: TIL, PO Box 74, Paddock Wood, Tonbridge, Kent TN12 6DW. £21.00 U.K., £24.70 Europe, Middle East £25.80, Far East £27.60, Rest of World £26.20, U.S.A. $45.00. Airmail rates available on request 0892 834783. Typesetting by Adtec Typographics, Britannia Court, Basildon, Essex. Tel: (0268) 591110. Printing by Southernprint Ltd. Sinclair QL World is published on the fourth Wednesday preceding cover date.

CONTENTS

MARCH 1990

NEXT MONTH

SOFTWARE FILE: HERE WE GO
An adventure in football – match postponed till next month.

FANCY STUFF PART 2
Now the longest-running 'next month' since the Artist of the Year, 1988.

QL SCENE

AN ALTERNATIVE SHOW

The First Alternative European Microcomputer Show will take place on Saturday April 28, 1990 at the Burgerhalle Seehein-Jugenhein, Bahnhofstrasse, Ortsteil Jugenheim, D-6104 Seeheim-Jugenheim, West Germany. The show, which is scheduled to run from 10am to 6pm, is expected to feature Sinclair, Acorn, Tandon, Cambridge Computers, Oric, Psion, MSX, Amstrad/Schneider and other 'alternative' machines. The promoter is Thornado Systems.

A show guide will be available on the door – priced DM1, 35 pence – and exhibitors are being encouraged to take advertisement pages in the A5-size guide for a modest fee. A profile of up to 200 words of each exhibitor will also be published free in the guide. Registration forms and information for exhibitors are available from the organiser.

Tickets will cost DM5, £2 in advance or DM7,50, £2.50 on the door. Children under 14 and any members of the Thornado Systems Club can enter free. Registration forms and further information can be obtained from **Thor Datensysteme, Kaninostrasse 25, D-6100 Darmstadt, West Germany.** A small map and travel information is included in the registration documents. Information about accommodation and hotel bookings is available on request.

CHECKMATE

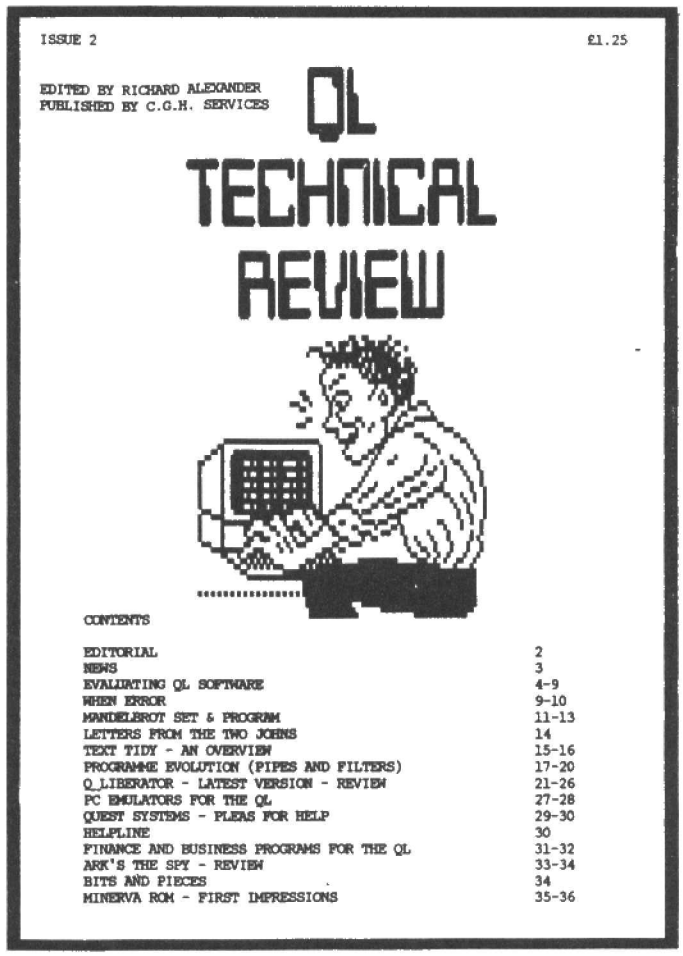
Joris Van Domme of Tron Software contacted *QL World* to inform readers that two bugs had appeared in *QL SuperDAM*, reviewed in last month's issue, as a result of late changes. The first bug relates to the screen dump facility and the second appears as a weakness in the computer's play when the human opponent is poised to take the king.

The bugs have been corrected in V2. Any queries about the program should be directed to **Joris Van Domme, Tron Software, Lindenlaan 10, 9370 Lebbeke, Belgium.**

Tron Software is apparently about to release a sub-directory system for floppy and hard discs, including such facilities as FIND and a fast copy command.

QL World

Despite constant talk of the 'declining QL market', QL World is showing a current firm sale figure of 12,400 – which is more than many specialist magazines in 'stable' markets. The QL market is not declining – it is refining.


ISSUE 2 £1.25

EDITED BY RICHARD ALEXANDER
PUBLISHED BY C.G.H. SERVICES

QL TECHNICAL REVIEW

CONTENTS

REVIEW

The second edition of *QL Technical Review* from CGH Services contains artcles on evaluating QL software, Mandelbrots and pipes, information about TexTidy from John Silk of PDQL, and a review of *Q-Liberator* V3,22 and *The Spy Editor*, as well as several shorter articles.

QL Technical Review costs £1.25 for one issue, £5 for four issues, from **CGH Services, Cwm Gwen Hall, Pencader, Dyfed, Cymru SA39 9HA. Tel: 055 934 574.**

SCOTLAND'S QL USER GROUP

The Scottish QL Users' Group has been in existence for six months, having been formed to provide a Scottish forum for the exchange of ideas concerning the QL. The group places an important emphasis on meetings, as there are few organised opportunities for QL users to get together north of the border.

The club's current venue is a village hall near Edinburgh but it also hopes to set up meetings in the West of Scotland if there is sufficient interest.

Activities at meetings include demonstrations and informal talks on all aspects of QL computing, as well as problem-solving and the occasional games-playing session. A monthly newsletter keeps members up-to-date on the activities of the group.

QL users wishing to join the group or request further information can contact **Alan Pemberton,** enclosing a stamped self-addressed envelope, at **65 Lingerwood Road, Newtonegrange, Midlothian, Scotland EH22 4QQ.**

OPEN CHANNEL

Open Channel is where you have the opportunity to voice your opinions in *Sinclair QL World*. Whether you want to ask for help with a technical problem, provide somebody with the answer, or just sound off about something which bothers you, write to: Open Channel, Sinclair QL World, Greencoat House, Francis Street, London SW1 1DG.

Horse

This is a simple program I wrote while working on a horse racing program. The program requires data about jockeys, trainers and courses to be input into a Psion program and then exported to SuperBasic.

As SuperBasic lacks an IMPORT command, I wrote my own. To use it, type:

```
IMPORT "mdv_file_exp"
TO "mdv2_file_dat"
```

Here is the listing. It has been IMPORTed from Basic, so there should be no errors:

```
100 DEFine PROCedure
IMPORT (source$,
destination$)
110 LOCal char, records,
get_races, header$
120 OPEN-IN #3, source$ :
INPUT #3, header$
130 DELETE destination$ :
OPEN_NEW #4,
destinations
140 records = 0 : PRINT #4,
"0000"
150 REPeat get_recs
160 char =
CODE(INKEY$(#3))
170 SELect ON char
180 =13 : records = records
+ 1
190 =44 : PRINT #4,
CHR$(10);
200 =26, 34 : REMark ignore
LF and quotes
210 =REMAINDER : PRINT
#4, CHR$(char);
220 END SELect
230 IF EOF(#3) · EXIT
get_races
240 END REPeat get_recs
250 SCROLL #4, 4 –
LEN(records), 42 : PRINT #4,
records
260 CLOSE #3 : CLOSE #4
270 END DEFine IMPORT
```

The file can be read using the INPUT command. The first INPUT will give the number of records in the file. I hope other readers will find this of use.

Andrew Veitch,
Edinburgh.

Caution

I am writing with a cautionary tale which may be of interest to readers. Having recently been given a Sinclair QL, I saw the Microfair as an excellent opportunity of checking the products and software available for the QL. The show had on offer a bewildering array of hardware and software to tempt the novice and more experienced user.

As a novice, I was looking for software of a practical nature and to that end bought two programs at very reasonable prices but, on returning home with them, I found to my dismay that they failed to run.

Although disappointed, I set out to trace the company which had sold me the software. I went through every journal and trade magazine I could but to no avail. I then tried my bank, which said it would forward a letter to the bank which drew the cheque against my account with a request that it was forwarded. That I did, and I received a letter to say that no account was held in that name and that the cheque had been passed through another account after being countersigned.

I have now no way of retrieving my money or getting the programs replaced, although both programs carry the Sinclair trade mark. It has now made me cautious about parting with my money. I am surprised that no check was made to see that the companies exhibiting at the show were bona fide and offered some comeback to the purchasers.

I offer this tale so that other users of this excellent machine are not caught the way I have been.

I. Braybrook,
Shooters Hill,
London SE18 3JF.

Editor's comment: This is a tricky situation and though it is more likely to catch people who are buying cut-price or second-hand goods, it also occurs in much bigger business ventures, including finance-related businesses.

There is no reason, in theory, why the bank could not trace that cheque to its eventual payee and permit an investigation into how the cheque had been obtained and from whom but very frequently banks will not co-operate with that kind of investigation, even where the police are involved, on grounds of confidentiality.

If the police are unable to obtain information concerning a suspected fraud it seems unlikely that a citizen will be able to do so. I have not heard a satisfactory explanation of why this situation exists but it certainly aids fraud. The law now requires banks to give information where profits from drug trafficking are suspected, so why not other criminal activities?

There is no absolute way of protecting oneself from deception. Crossing all cheques "account payee only", however, means that the cheque must be paid into the account of the person or organisation to which it is made out. That offers some security. Asking for a receipt with a name and address on it – better still if there is a VAT number but this will not

Editor's notebook

First, you will have noticed that *QL World* appears a week later than usual. Please do not blame the printer, the distributor or the postman. It is part of the plan. I have been on holiday and decided in advance that I should look at the magazine before it went to press. The April issue will also be a week late, after which we shall be back to normal.

Rumours have reached us that somebody has been claiming that the *QL World* circulation is roughly half the real figure. Our current firm sale figure is 12,400 copies – not bad for a 'dead' machine. Somebody – who knows why? – has been encouraging the idea that the reduced number of pages in the last two years means a decline in the amount of editorial material readers are getting. That is not so; *QL World* contains as much editorial matter as it did a year ago and compares with issues two years old. What do readers think?

QL Sub has now contacted its members to explain its current position and ask for more financial support. The concern felt by some observers that it would not be able to meet its projected level of service on current subscriptions would seem to derive some support from this. Members and prospective members may now have to decide whether they will back Sub or turn to one of the clubs, like Quanta or QLAF, which offer perhaps a less ambitious but more regular service.

apply to smaller dealers – also gives some proof of authenticity. If in doubt, see if you can obtain permission to run the program; there are usually QLs up and running somewhere nearby.

If the dealer is wary, of course, it may be that he is also afraid of being taken advantage of. There is no way that show organisers can be certain that every dealer who takes a table is bona fide, as any individual is allowed by law to sell goods.

Inverted

I was surprised that you were interested in my inverted QL. I hope that the pictures this time are more suitable for, as you know, originally I sent them in as an addition to the 'sawn-off QL' joke.

The reason for the amputation, as you can see, was the limited area I had in which to set up the QL permanently. The QL PCB is mounted, upside-down, directly underneath the keyboard on a plank of wood between my bedside cabinet and the wall. In this way I still have the use of the Microdrives if required. The cable and connectors were all supplied by Maplin, the electronics component suppliers, and are:

1 x 25-way IDC plug, number FV81C @ £2.95
1 x 25-way IDC socket, number FV82D @ £3.25
1 x edge connector, number FL83E @ 98p
1 x length of 26-way IDC cable, number XR 75S, as required @ 34p per foot

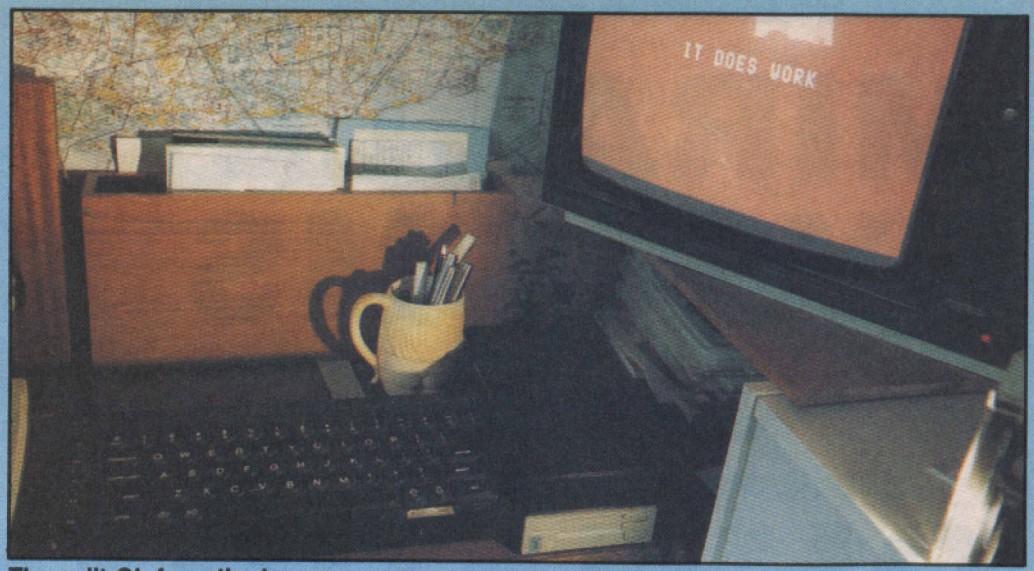
Add 15 percent VAT to the total.

Remove the whole circuit board from the case, being very careful when removing the keyboard plastic cables – films – from the sockets. Solder the IDC 26-way ribbon cable to the bottom of the circuit board where the 9- and 11-pin keyboard sockets are attached, taking all the usual precautions to see that the board does not become too hot or suffer high voltages. Connect the wires, which are red and black on mine, for the ON led on the keyboard to the IDC cable.

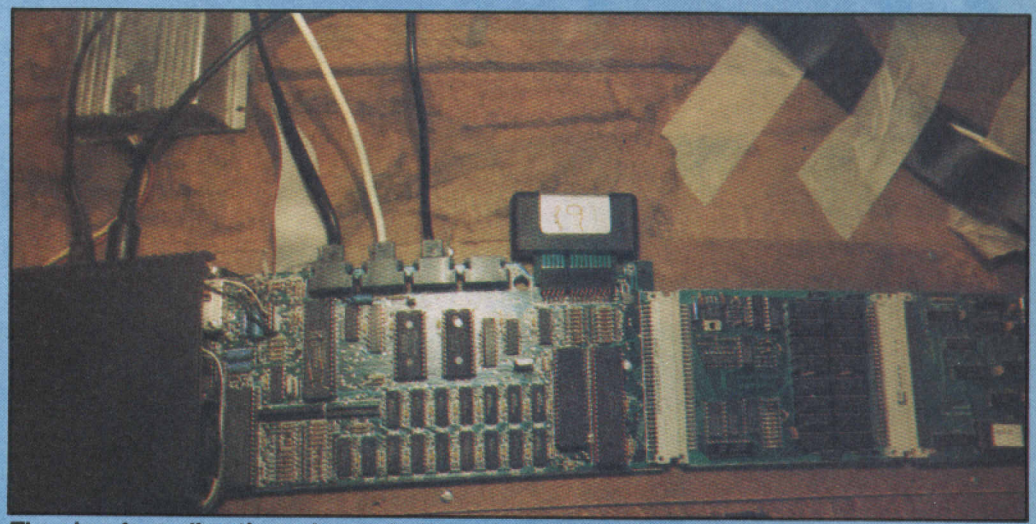
Attach the 25-way plug to this, keeping the IDC cable as short as is practicable to reach wherever you are to mount the removed PCB. Now we approach the point of no return. Cut the Microdrive housing from the keyboard – I cut mine with a junior hacksaw blade – and remove the separator between the ser1 and ser2 outlets to allow the cable to go through or to mount the socket.

Cut the edge connector socket into two pieces to suit the 9 and 11 contacts from the keyboard. To those, solder a short piece of IDC cable, reversing one of the pieces to match the film from the keyboard. To the other end attach the 25-way socket and also the led wires and so on.

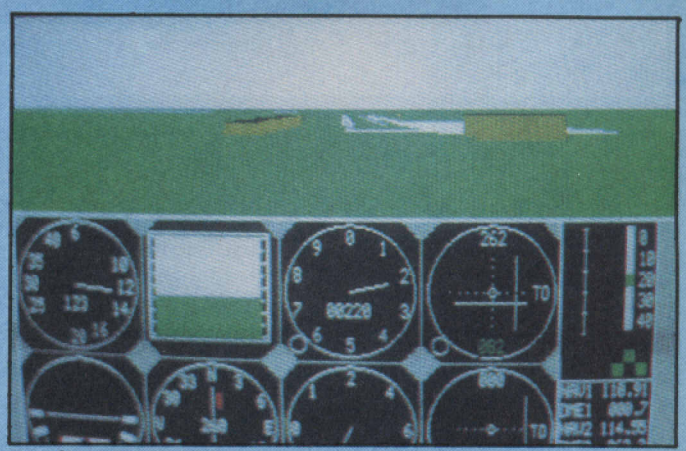
Slide the keyboard films onto the made-up sockets along with a piece of thin plastic or card; do not be too brutal as you can easily damage the film. A strip of Lasovic tape will prevent any movement once inserted.



The split QL from the top.



The view from directly underneath: the pcb suspended from a shelf.



Flightdeck in 3-D.

The one thing I did not move was the re-set button but I think there are enough spare wires in the IDC if anyone needs to do this.

The third picture I have enclosed is from Berard Denchfield's excellent *Flightdeck* and it is my impression of Gatwick on approach on 262 left, proving that if you want vertical 3D he has given you the facility to do it, though it slows the picture a little.

The main uses for my set-up are the usual accounts and letters, programs from *QL World* and flying round Great Britain courtesy of Flightdeck when I have the time.

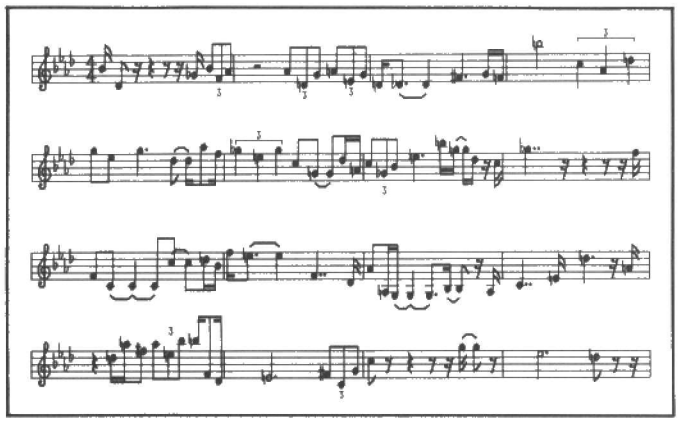
S.J. Parker,
Lindfield,
Haywards Heath,
West Sussex.

SOFTWARE FILE

MUSI-QL

INFORMATION
Product: MusiQL – music notation printer
Supplier: I. N. Back, 1 The Chase, London SW4 0NP
Price: ??

Mike Lloyd presents another view of the professional-quality notation program reviewed last month.



Some months ago a regular reader of *Sinclair QL World* sent a disc full of music-writing programs he considered might be suitable for including in Microdrive Exchange. Sadly, the package was too big to be sold on a single Microdrive and, in any event, its potential was so great that our advice to the author was to release the programs professionally. Dr Ian Back has since identified the most commercial elements of the programs, modified and improved them and launched himself as a small, specialist software house.

Commercial

I mention this in detail because in the QL market, above any other, the way is open for competent amateur programmers to produce quality commercial software for a discerning public. Try it in the Atari 520 market, for instance, and you will be swamped among the thousands of small advertisements. One reason why the QL is such a good base for software is that its compiled Basic is a supremely competent language. *MusiQL*, the program being reviewed, is one example of the many successful pieces of software to be compiled by SuperCharge, itself just one of three quality compilers QL users can choose.

MusiQL requires a QL with up to 256K of memory expansion connected to an Epson-compatible printer. Its objective is to to produce printed music and so it can be described as a kind of musical word processor. The package comprises three programs – one to write music, the second to control printing and the third to produce blank manuscript paper for handwritten music. The last option alone should help users recoup the cost of the program through not needing to buy expensive pads of manuscript paper. One thing it does not do is play music; for that you will still need human beings and musical instruments.

To begin writing a tune in MusiQL the working copy of the program is placed in the default drive of a re-set QL and the F1 key is pressed. The Tune option is picked from the menu, its code is loaded from the disc and the Tune sub-menu is displayed. This menu is operated by pressing the first character of the required option, whereas in other menus numbers are pressed. The options are named intelligently so that their purpose is obvious to anyone who has read the Quill-based manual.

A beginner will start with the New option which begins by posing questions of the new tune title, clef, time signature and key signature. Those matters resolved satisfactorily, the screen displays a length of musical stave complete with clef, signature and key written in and supported by an aide memoire for all the input options which are available. Music is entered one bar at a time with a maximum of 32 notes or rests per bar.

Piano

There is no easy way to use a QWERTY keyboard to type-in musical notes. It is just as awkward if the roles are reversed; imagine typing a letter on a piano. It is therefore no criticism of MusiQL that it is sometimes cumbersome to use.

MusiQL supports nine octaves, 108 notes in all, from the deepest bass to the highest treble. Each octave is given a number from 0 to 8, in line with standard Midi notation. The 12 notes in each octave are distinguished by the eight letters of the octave scale, with intervening semitones identified by pressing either the up arrow for sharps or the down arrow for flats.

Written notes convey duration as well as pitch and so MusiQL requires each note to be followed by a mnemonic representing its length – "Q" for a quaver, "M" for a dotted minim, and so on. A Middle C lasting for half a beat would therefore be entered as "C5Q". I soon tired of having to enter the octave number and would welcome a little more intelligence from the program so that octave values need be entered only when the "current" octave is left.

The most disappointing feature is the unnecessary differentiation between input mode and edit mode. It is rather like Quill being re-modelled so that it has a mode in which deleting characters is forbidden and the cursor can never retrace its steps. Worse, if an "end of bar" code is inserted in the incorrect place MusicQL forces you to re-type all the notes in the bar, presumably as a punishment. That was frustrating with crotchets but it must be murderous with semi-quavers. Because of this peculiar piece of pedantry it is also impossible to type-in the first two notes of tunes which do not begin on the first beat of the bar.

The cure seems to be to dispense with the input mode in favour of the edit mode, teach the program to place bars on the stave for itself, and to cope with miscalculations of note values in a more friendly manner. Perhaps version two will address those points.

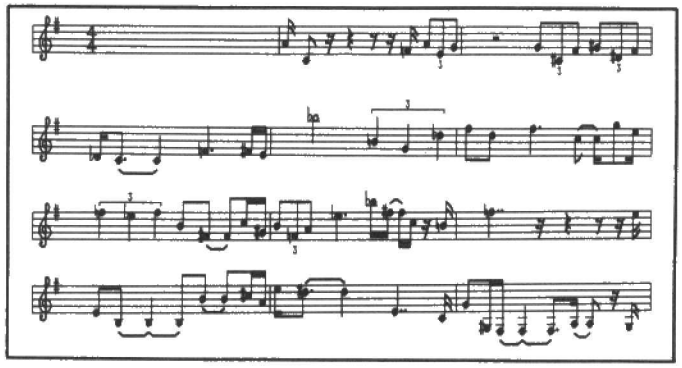
The edit mode was much

more friendly, allowing bars to be copied from anywhere in the manuscript, notes to be changed, bars to be re-written and the tune to be extended by adding new bars to the end of those entered in the input mode. A maximum limit of 256 bars is imposed for any single tune but a magnum opus could be spread across several files.

The Tunewriter module does not recognise multi-stave music. A determined user could force it to print tunes with linked treble and bass staves but the process would be frustrating and difficult. Version one is more naturally suited to writing music for instruments which play on a single stave.

Once entered and edited, tunes can be viewed in any key and in any pitch. Selecting a new key forces the appropriate sharps or flats to be displayed at the beginning of each line. Intelligent decisions are made about whether notes such as F sharp should really be G flat and there is provision for users to over-ride the program's choice.

The file management options are conventional and competently error-trapped.



Old tunes can be loaded and edited and up to 256 tunes can be saved to a single disc or Microdrive. Directory listings restrict themselves to files with the MusiQL filename suffix. Before a tune can be printed it must first be saved and the second executable program in the MusiQL suite overlaid into memory.

In the Tunewriter module previously-saved tunes can be loaded into memory one at a time. The key and the pitch can be changed so that a tune written in G can be printed-out in E flat it the user desires. The density of music can be altered by selecting two, three or four bars per line. Settings for single sheet, continuous stationery, draft mode and full quality mode are available. In practice, draft mode was of a good quality and was twice as fast as the full quality mode. Ties, triplets, joined tails and 8va markings are taken in the program's stride.

Strength

The professional look of the printed output is the real strength of this program, as can be seen from the examples reproduced. Its ability to represent, with great clarity, highly-complex music is clear to see and a tribute to the designer. Standard musical notation has all the elegance, decoration and grace of the baroque music it was developed to represent.

It is unsuited to translation into a character set similar to the English alphabet. The printer therefore works in graphics mode and the program "draws" each line of music in memory before sending information to the printer. The painstaking care taken to ensure that the printouts cope with most musical eventualities shines through the design of each symbol.

The final product supplied with MusiQL is called Manuscript; it produces blank manuscript paper, pre-marked with stave and key, for handwritten music. The output is four bars per line and staves can be joined in pairs, threes, fours and sixes, any number of which can be given bass clefs.

MusiQL is a very impressive product, especially as it is the first commercial program sold by its designer. The weaknesses in its communication with the user are more than offset by the quality of its output and regular music writers can safely buy MusiQL and know that they can make good use of it. If the demand is sufficient the product could develop further and be joined by other related utilities.

SOFTWARE FILE

SB Mon

Mike Lloyd finds a fine-tuner.

Some programmers take the search for ever-improving levels of code efficiency to extraordinary lengths. One such programmer is the irrepressible Chas Dillon, responsible for *The Editor* and once a member of the team which produced the Digital Precision Turbo Basic compiler. Dillon is a fastidious codesmith whose products set exemplary standards for efficiency, reliability and functionality. His latest offering is a programmers' utility, *SBMon*, released through the Midlands software house PDQL.

SBMon is an unusual offering which seems to have been developed to meet Dillon's need for an objective method of identifying which parts of his programs should be tuned for maximum performance. It runs as a background task when a SuperBasic program is running and, at intervals specified by the programmer, records which line the SuperBasic interpreter is examining.

The object is to identify those parts of a program which are called most frequently and which would benefit most from fine-tuning.

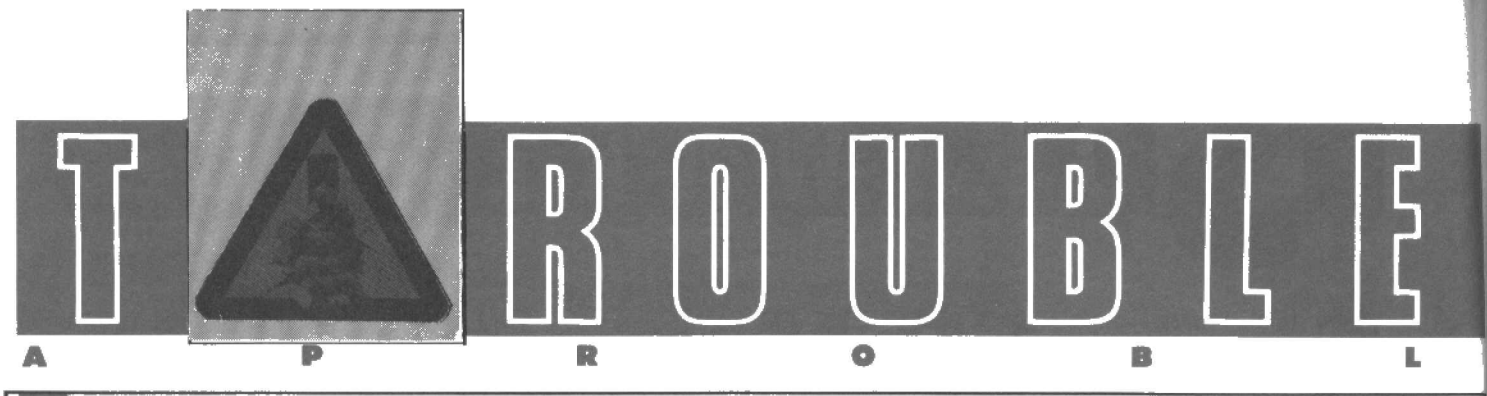
Programs are divided under the user's control into a maximum of 300 "analysis cells" which potentially hold the same number of program lines. A declared cell interval might contain only three or four widely-spaced program lines. For detailed work programmers can restrict the range of lines being monitored and can thereby allocate one program line to each analysis cell.

Output can be directed to the screen, a file or the printer and comprises columns for line numbers, the frequency with which they occur and two calculations of percentages, one including and the other excluding time in which Basic was idle.

While acknowledging that the utility has been executed to a very high standard I am not convinced that it is particularly useful. Most programmers are aware of which parts of their programs are called most frequently and where there is doubt the temporary inclusion of a well-sited BEEP command can be most instructive. The statistics gathered by SBMon can be misleading if line numbers are spaced unevenly or if the program waits for keyboard or file input for any length of time.

Nevertheless, SBMon might uncover a key routine in very large programs which would otherwise be missed. It could also prove its value when a program is undergoing pre-release testing by volunteers in the programmer's absence. The testers may say they spent most time in one particular area but SBMon may reveal otherwise.

If you write large commercial programs and use volunteers to test them, SBMon could be worth trying; otherwise it is a luxury of limited worth.



Bryan Davies takes a long hard look at the world of QL software.

It has been a notable feature of the last few years that, as suppliers fell by the wayside, the software and hardware really started to come good. Be assured that there are still more good things to come. There are projects under way now which will fill some, at least, of the obvious gaps in the QL armoury. No one in his right mind embarks on a long development project without having done some research into the potential market for the end product; there must be sufficient potential buyers to make the developer a reasonable amount of money.

Some software writers to whom I speak have believed for some years that money is not available in the QL market in sufficient amounts to feed their interest or stomachs. It is surprising to discuss figures and realise what a gap there is between the optimistic and pessimisitic ends of the scale. Two writers for whom many people have considerable respect have told me that they think potential sales of even good, major programs are now no more than 1,000-2,000 copies.

In contrast, one supplier appears to feel that these figures are more like a guaranteed minimum and the maximum is much higher. We are talking about the border between breaking even and making reasonable money. If nothing can be sold in greater volume than than 1,000 copies, only those writers and suppliers producing or distributing several major programs can survive by working with the QL alone.

Nobody will commission a survey to determine the buying habits of QL users but think of a few statistics. Sinclair was responsible – directly or indirectly – for the production of sufficient parts for perhaps, 250,000 QLs. There is no point in nitpicking. If the figure is truly only 150,000, it does not make much difference to the argument.

It seems to be accepted that only about 150,000 QL systems were completed; although some were completed after Sinclair ceased to be directly involved, there have not been enough of them to alter the total significantly. Roughly half the total went out of the U.K. and were spread widely throughout the world. No doubt many are now covered by dust in cupboards but there is little doubt there are a few thousand still being used by people who are active enough to get their names on mailing lists in various places.

The User Group, Quanta, has a membership approaching 2,000; *QL World* has a circulation several times that; SUB claimed approaching 1,000 subscribers; and at least one supplier has a database of around 30,000 names. In the usual way of things, there are many more people involved than those whose names are listed on database but it seems reasonable to suppose that there are 20,000–40,000 people who are potential buyers of QL products, with perhaps 5,000–10,000 of them being fairly active.

The relatively high prices asked by sellers of used QLs and by suppliers of new ones give further support to the feeling that the QL scene is an active one; complete used systems appear to be fetching prices comparable with those of the cheapest new PCs.

Prices

In the long run, a typical user may spend almost as much on software as on hardware. Those buying a QL now at £100 or less will soon exceed that expenditure on programs. That is one of the main merits of buying a QL now; there is a fairly wide range of good software available at prices which are generally below those associated with competitive computers. A cheap PC may look like a good deal but paying as much for a word processing program as for the computer system soon dispels any notion of cheapness.

Lack of sales for software should not automatically be attributed by the writers to lack of potential buyers. Selling is not a straightforward matter; if it were, would such a vast amount be spent on advertising and promotion in general? Suppliers with a wide range of good products will take plenty of orders, provided they advertise sensibly and regularly. In this context, advertising includes making appearances at fairs and meetings for QL users, largely to overcome the distrust many users have, with good reason, of buying by mail order.

Users and buyers are not the same of course. The bane of a software supplier's life is piracy and some people believe that accounts for as much as 50 percent of the program copies in use. It is said to be more prevalent for QL titles in countries outside the U.K. and that is at least partly due to ineffective distribution of "the real thing" in other countries.

One line with which Schön has had success is add-on keyboards. While the desire to get away from the QL keyboard makes little sense to me, it clearly exists strongly for a significant number of users. Even though they may not be advertised, the same keyboards are still available from suppliers such as PDQL. It is also possible to get a keyboard interface to allow standard IBM-pattern keyboards to be connected to the QL.

There has been a steady but, presumably, small demand for PC-style cases to house the QL plus a few add-on units. The ones which were available left more than a little to be desired, so far as the non-DIY user was concerned. Are there any left on the market or about to be launched?

To my mind, the piece of hardware most conspicuous by its absence is a mouse. It would be easy to conclude that there is no sensible way of attaching a mouse to the QL, since there has never to the best of my knowledge been a "standard mouse" for it. It has not been possible to buy a mouse which could be fitted easily by most users and be useful with the obvious, major programs. I think you can still obtain the Ice mouse from Transform International but it is a dedicated device, working only with the Ice front-end firmware, Artice, and a few other small routines.

The Smiling Mouse showed promise of being fairly inexpensive and usable with programs such as *Front Page* and *Desktop Publisher* but the development of the interface software and the marketing of the package were not handled well and it disappeared from the scene. More or less the same story applied to the Giga Mouse from ABC. Qjump offered a mouse, presumably intended for use with its own software; whether or not it is usable with graphics or DTP programs such as *EyeQ, Professional Publisher* and *Page Designer* I do not know.

In the absence of any advertising from Qjump recently it will be uncertain to potential buyers whether or not this mouse is available. All this adds up to at

SHOOTER

EMSOLVED

least three hardware items which are in need of a producer and supplier and some advertising to bring them to the attention of users.

One should read the small print now and then. I knew that *text*[87] version 3.00 would incorporate several useful improvements but it had not occurred to me that one of them would be a spell-checking facility. The recent advertisements note this, without making any special point of it. Surely the incorporation of a spell-checker in a word processing program for the first time on the QL deserves a little more noise from the suppliers. The checker should operate rather faster than the *SpellBound-FileBound* combination, plus having the marked time and space advantage of not requiring stepping out of the WP program to do the checking. I hope it will be possible to comment on hands-on experience with the new version in next month's issue.

There must be many users who feel that a PC emulator on the QL cannot be used for serious work because software emulation is so slow. There is no doubt that lack of speed is a big disadvantage but it is wrong to think it prevents you doing sizeable jobs. As a test, I took a 20,000-word document – almost 200KB in size – directly from *WordPerfect 5.0* on a PC/AT and loaded it into the same program on the QL, using *PC Conqueror*. It was by no means a five-minute process; it took some time to determine the best approach to getting such a large document loaded and the loading was not fast. Much of the time spent was no reflection on either the QL or Conqueror – it is not only Quill which uses overflow files for large documents. The current generation of PC programs really cannot be used sensibly on computers which do not have a hard disc drive and this factor tends to make the QL-plus-emulator look unjustifiably slow.

What mattered most, though, was that the 52-page document loaded properly and was displayed equally as well as the original at the AT standing alongside the QL. The boxes incorporated in the document for subsequent pasting-in of pictures were there, shown clearly, and the Print Preview function worked properly, although naturally not so clear as on an EGA screen.

The real test is always printing and there was no problem there either. The output was correct and did not take as long as might have been expected. Editing of documents can be done at a reasonable pace but you need to avoid using commands as far as possible, because they take proportionately longer than text entry.

Discs can be an expensive item and there is the usual potential for losing money when buying by mail order from unknown suppliers. Recently I needed 3.5in. discs and, having seen no advertisements from my previous supplier Athena, had to risk buying from a new one. The list of possibles numbered about half a dozen, selected on the basis of lowest price, some kind of replacement promise for faulty discs, acceptance of credit cards and the look of the advertisement.

Credit Card

The last factor is obviously not the most precise of tests but I certainly dismissed one supplier purely because the advertisement suggested a rip-off to me. Credit card payment is absolutely essential if you want any kind of protection against losing your money. A bank will not help at all if your cheque is cashed and the goods do not arrive but a credit card company is likely to take more interest in your case.

As mentioned in a previous article, £25 I lost to mail order company for printer ribbons which never materialised was refunded to me by Access. Price varies more than one might expect and is confused by offers of "free" disc boxes with larger orders. Buying 20 at a time, the best prices for 3.5in. discs ranged about 66-90 pence each. Although unbranded discs with no guarantee can be satisfactory to most users, it seems desirable to pay a little extra and get some form of promise of replacement, such as "all discs 100 percent certified and guaranteed".

Few of my discs are branded and few have given any trouble through the years. The supplier I chose finally – **Manor Court Supplies**, see information – had the lowest effective price – £20 (total) for 20 discs plus an 80-disc storage box but without labels. Depending on how one values a box that equates to something between 66 and 83 pence per disc, which I think acceptable.

The test of a 3.5in disc is, for me, whether or not it will format to 1.44MB on the AT. The fact that the discs are sold as 720KB capacity, 135tpi does not prevent them being used on some drives at the higher capacity. The 20 bought this time all gave the full 1.44MB first time without error. That implies that they will be satisfactory used on the QL at 720KB. Having been formatting unbranded discs to 1.44MB for more than a year I have yet to find any significant problem with them. There have been two or three instances of discs giving less than the full capacity and those are now used only at 720KB; the few instances of disc errors appeared to be related to software problems, not to the discs.

The letter from **James McGreehin** in the February issue appears to have got out of sequence in the multi-way correspondence chain on the subject of printing with *graFix*. Digital Precision is not the producer of this program – PDQL is. It was apparently supplied together with Special Edition Desktop Publisher and Professional Publisher but not with the basic Desktop Publisher, which is presumably how the confusion arose.

Both PDQL and DP state that the Serial 8056 printer is unsuitable for use in DTP work but there appears to be some problem related both to printing of Desktop and/or Professional Publisher files by graFix and to the symmetry of printouts from the program in general. The problems should be sorted out by now and a new version of graFix should be available; it will include multipass printing.

David Owen asked about the current status of the Italian supplier **SPEM** on the QL scene. It has always been a little difficult to find who handles products and one has to assume that it ceased trying to find a suitable U.K. distributor some time ago. The base address I have is given in the Information box; if anyone has up-to-date news of its interest in the QL and what products it has on offer, we would like to receive it.

INFORMATION

Discs: Manor Court Supplies Ltd,
Dept. PCW
1 Glen Celyn House,
Penybont,
Llandrindod Wells,
Powys LD1 58Y
Tel: 0597 87784

SPEM
Via Ponchielli 26c,
10154 Torino,
Italy

SUPER BASIC

Mike Lloyd presents a definitive description of the QL character set.

The QL character set is unusual in a number of respects. Its characters are not square but rectangular; their size and spacing can be varied using SuperBasic commands; non-printing characters are depicted by a standard pattern and there are not one but two character set definitions in the QL memory. With these potential complications and a complete lack of guidance in the User Guide it is not surprising that programmers hesitate to experiment with changes to the standard QL fonts.

Mechanics

The mechanics behind printing a character on the screen are simple. Each character is given a code number from 0 to 255 from which the computer can calculate where in its memory to search for the character definition. The definitions comprise a number of bytes which make sense only when written in binary. Each binary digit represents a screen pixel, a "1" being an INK pixel and a "0" being a PAPER pixel. The QL ROM contains a routine to carry-out this translation process and paint the pixels at the appropriate point on the screen.

The memory areas in which character definitions are stored are called fonts. The code numbers given to the characters are, as most QL users know, standardised on the ASCII set which determines that the alphabet in capital letters begins at ASCII code 65. Codes below 32 are "non-printing" codes used to represent linefeeds, tabulation, formfeeds, backspacing and so on. Codes above 127, the copyright symbol, can be assigned to different characters according to the needs of the particular computer manufacturer. The QL has a large variety of additional characters stored from ASCII code 128 to 191.

If users wish to create new character sets or modify selected characters from the standard font they need to allocate RAM space for the new font, copy into it any standard characters they wish to retain, insert new definitions for the character codes which are to be changed and then tell the computer to ignore the standard font and access the new font instead. Before any of that can be attempted an understanding of the layout of the standard fonts is essential.

In the JS release of the QL ROM the first font begins at memory location 44442 and contains definitions of every character from ASCII code 32, the space, to ASCII code 127, the copyright symbol. The first byte in the font is one less than the ASCII code of the first printable character to be defined in it, i.e., 31. The second byte represents how many characters the font contains, i.e., 96.

Each character has nine rows of pixels, although some of them may be blank. Nine bytes are reserved for each character definition, one for each pixel row. Most computers use characters which are eight pixels wide but the QL has its characters only five pixels wide. That means that the first bit and the last two bits of each byte are ignored. The QL cares not whether the final two bits are ones or zeros but occasionally it will corrupt the screen if the first bit is anything but zero.

The first character definition in each font is a standard pattern, the familiar chequerboard, which is used whenever an ASCII code which lies beyond the range of the font is printed. The start of each character definition can be calculated by finding the address of the first byte of the font, adding 11 to it to skip over the information bytes and the chequerboard pattern and then adding a number nine times the difference between the ASCII code of the required character and 32, the ASCII code of the first printable character. The total length of the first font is 875 bytes but an extra null byte is added so that the next font can begin on an even-numbered address.

The second ROM font is organised in the same way as the first with two bytes of information and a default pattern – the chequerboard again – followed by character definitions from ASCII code 128 to 192.

The number of characters in each font is not fixed. Programmers can declare a single font containing all the printing characters from ASCII code 32 to 191 and even include non-printing codes. Each font must begin with the two information bytes set correctly, followed by the pattern to be used for character codes beyond the range of the font.

Speedscreen, Lightning and the QJump *Super Toolkit II* include additional SuperBasic commands which allow new fonts to be assigned to windows. Programmers without those facilities, or those writing programs for users who may not have them, must write their own routines to produce the same results.

This month's program is a collection of font-designing utilities which combine simplicity with flexibility. New characters can be designed and described in an easy way and converted into either of two formats for saving and incorporation into other programs. To keep things simple, only the first of the standard fonts is used and it is kept to its original range of ASCII code 32 to 127.

Reserved

The first listing takes a copy of the standard font and places it in a reserved area of memory. Four bytes of data are transferred at a time to reduce the number of iterations and thus accelerate the execution time. It has been assumed that the standard font begins at memory address 44442, as it does in JS ROMs. If you own a different ROM – discover that by typing-in *PRINT VER$* – a different address might be needed. For other QL ROMs, type-in the Wbase function – listing five – and then use the following command:

```
PRINT PEEK_L (Wbase + 42)
```

To whatever address this points will be the first byte of the first standard font. The value 44442 should be replaced by this new value in lines 115, 305 and 5010.

The Wbase function assumes that the main Qdos reference table is located immediately after the screen map beginning at address 163840. It is very rarely displaced but it is possible that strange peripherals or software might make it inaccurate. In those circumstances you should consult the documentation of whatever is causing the problem or use a proprietary toolkit.

Copies

The Begin procedure copies the standard font into a reserved area of RAM where it can be manipulated. Before proceeding further some new character definitions need to be designed. Listing 10 contains the definitions of three example characters. It is not necessary to re-define every character in the font or to select groups of contiguous characters for redesigning. Each definition contains 10 pieces of information, consisting of the character being defined followed by nine strings which describe its shape.

PAPER pixels are represented by zeros or spaces. Any other characters are assumed to represent INK pixels. Blank rows can be shown by a null string. Characters can be designed first on graph paper and then translated into DATA statements or you can design directly on the screen by typing-in and modifying DATA statements. The appearances of the example characters can be seen clearly from listing 10. The last DATA statement can be any string with more than one character in it but a meaningful message such as "END" is recommended.

It is important to ensure that exactly nine strings exist for each character; any more or less will force the display routine to finish prematurely with the offending data string printed on the screen.

Listing two controls the conversion of DATA strings into bytes and places them into the font. As each character is defined it is printed alongside the standard character shape in Window#1. Much of the work of this procedure is sub-contracted to subordinate routines described later.

The *Cbase* variable holds the first byte of the character being defined. A FOR..NEXT loop cycles once for each of the nine rows of the character definition. Nine DATA strings are read, converted to decimal values and POKEd into the correct memory addresses. The new character font is then selected with the NewChars procedure and the design is printed in Window#1.

Listings three, four and five control which font is used in which window. Each window has a block of information associated with it to retain details of such things as the current INK and PAPER colours, the cursor location, the border size, and so on. The start addresses of the two fonts associated with each window are four-byte values – long words – offset 42 and 46 bytes from the start of the data block.

These information blocks can be shunted round in memory by Qdos and so they are located via a pointer system. The long word stored at location 163960 points to a table of channel pointers, also

Listing 1

```
100 DEFine PROCedure Begin
105 LOCal X, Cbase
110 PRINT#0, "Initialising ..."
115 Base = RESPR(900): Cbase = 44442
120 FOR X = 0 TO 866 STEP 4
125   POKE_L Base+X, PEEK_L(Cbase+X)
130 END FOR X
135 PRINT#0; "Complete."
140 END DEFine Begin
```

Listing 2

```
200 DEFine PROCedure Display
205 LOCal loop, C$, Cbase, Cbyte$, Cline
210 RESTORE 1000: CSIZE 2, 0: CLS
215 REPeat loop
220   READ C$: QLChars(1): PRINT C$; " = ";
225   IF LEN(C$) > 1: RETurn
230   Cbase = Base -277 +CODE(C$) *9
235   FOR Cline = Cbase TO Cbase +8
240     READ Cbyte$: POKE Cline, BinDec(Cbyte$)
245   END FOR Cline
250   NewChars(1): PRINT C$
255 END REPeat loop
260 END DEFine Display
```

Listing 3

```
300 DEFine PROCedure QLChars (Screen)
305 POKE_L Wbase(Screen) +42, 44442
310 END DEFine QLChars
```

Listing 4

```
400 DEFine PROCedure NewChars (Screen)
405 POKE_L Wbase(Screen) + 42, Base
410 END DEFine NewChars
```

Listing 5

```
500 DEFine FuNction  Wbase (Screen)
505 RETurn PEEK_L(PEEK_L(163960) + 4*Screen)
510 END DEFine Wbase
```

```
Listing 6

600 DEFine FuNction  BinDec (A$)
605 LOCal Ans, Bit
610 IF LEN(A$) = 0: RETurn 0
615 Ans = 0
620 FOR Bit = 1 TO 5
625   IF Bit > LEN(A$): EXIT Bit
630   BitCode = CODE(A$(Bit))
635   IF NOT A$(Bit) INSTR " 0"
640     Ans = Ans + 2^(7-Bit)
645   END IF
650 END FOR Bit
655 RETurn Ans
660 END DEFine BinDec
```

```
Listing 7

700 DEFine PROCedure SaveChars
705 PRINT#0;"Name Device and File :-"
710 INPUT#0;File$: SBYTES File$, Base, 900
715 END DEFine SaveChars
```

```
Listing 8

800 DEFine PROCedure LoadChars
805 PRINT#0; "Name Device and File :-"
810 INPUT#0; File$
815 Base = RESPR(900): LBYTES File$, Base
820 END DEFine LoadChars
```

long words. The start address of each channel block is read from this table, as listing five demonstrates. Values passed to this function must be channel numbers linked to windows and not to files or to the printer.

Listing three pokes the address of the standard font into the appropriate position in the data block for a window, whereas listing four pokes the address of the new font in RAM. Beware of using non-alphabetical fonts for the command and listing consoles – Window#0 and Window#2 – because listings, commands and error messages become unreadable.

Listing six is where the DATA strings are processed into decimal values. The function begins by disposing of a special case by returning zero if a null string is passed to it. The FOR..NEXT declaration in line 620 reflects the fact that only five bits of each byte are used in character designs.

The design of the function makes it impossible for it to crash. All valid string lengths are catered for, with only the first five characters being recognised in over-long strings. Spaces and zeros are treated as binary 0 and any other character is deemed to be a binary 1. This type of implicit error-trapping is not always possible but when it can be achieved it lends a satisfying elegance and robustness to a program.

Rather than carry-out the foregoing process each time a program uses a new character set, there are two methods of introducing previously-designed character sets quickly to a program. The fastest option is to save the font area to a file and reload it when required. Listing seven creates a command which, when typed-in, prompts for a file. Listing eight reloads the file contents into a new reserved RAM area identified by the variable Base which can then be used by the QLChars and NewChars procedures to switch between fonts as before.

Disadvantage

This method of font storage is quick and easy but it has its disadvantages. The second storage method creates its own listings which can be merged with other programs. This method is ideal for games which re-define only a few characters.

Listing nine creates a file containing a procedure for installing new characters and the DATA statements which define them. These DATA statements contain decimal values rather than the bulky pseudo-binary strings used in the main program. You may wish to change the filename suggested in line 915. Lines 5000 to 5075 of this program are then listed to it. The high line numbers ensure that the listing is unlikely to clash with existing lines in the programs with which it will be merged.

```
Listing 9

900 DEFine PROCedure WriteOut
905 LOCal loop, Cbase, Cline, Cbyte$
910 LOCal Linum, D$, Q$, C$
915 RESTORE 1000: OPEN_NEW#3, flp1_Char_tmp
920 LIST#3, 5000 TO 5099
925 PRINT#3; "5100 REMark New Characters"
930 Q$ = "'": C$ = ", ": Linum = 5102
935 REPeat loop
940   READ D$: PRINT "Copying "; D$
945   PRINT#3, Linum;" DATA "; Q$; D$; Q$;
950   IF LEN(D$) > 1: EXIT loop
955   Cbase = Base -277 +CODE(D$) *9
960   FOR Cline = Cbase TO Cbase + 8
965     READ Cbyte$: PRINT#3, C$;PEEK(Cline);
970   END FOR Cline
975   PRINT#3: Linum = Linum + 5
980 END REPeat loop
985 PRINT#3: CLOSE#3
990 END DEFine WriteOut
```

```
Listing 10

1000 REMark Character Definitions
1002 DATA                "!"
1004 DATA "1"
1006 DATA "11"
1008 DATA "111"
1010 DATA "1111"
1012 DATA "11111"
1014 DATA "1111 "
1016 DATA "111  "
1018 DATA "11"
1020 DATA "1"
1022 DATA                "#"
1024 DATA " *** "
1026 DATA "* * *"
1028 DATA " * * "
1030 DATA " ***"
1032 DATA "*****"
1034 DATA "*****"
1036 DATA " *** "
1038 DATA " * * "
1040 DATA " * * "
1042 DATA                "$"
1044 DATA ""
1046 DATA ""
1048 DATA "  $"
1050 DATA " $ $"
1052 DATA "$   $"
1054 DATA " $ $"
1056 DATA "  $"
1058 DATA ""
1060 DATA ""
1062 DATA "END"
```

The routine then creates a number of DATA statements beginning at line number 5102. A loop is repeated once for each character definition. The standard ASCII character is copied in quotes as the first item in a DATA statement. A FOR..NEXT loop then cycles nine times, once for each line of pixels in the character definition.

The routine PEEKs data from the reserved memory area to avoid having to convert the DATA strings into decimal values once more. The DATA strings are read and discarded to keep the READ cursor up-to-date as the process moves from character to character. At the end of each DATA statement a newline is printed and the line number is incremented by five.

Each program line created consists of a line number, the DATA keyword, a single-character string identifying the character being defined and eight decimal values between 0 and 255 interspersed by commas. When the final DATA item, "END", is detected it is written to the file and then the file is closed.

To use this feature, first design and incorporate a few new characters into a font and then type-in the command Write-Out. The file will be created in the way described. Next, load the program which will make use of the new characters and MERGE the char_tmp file with it. At the beginning of the program include the command WriteIn which will copy the standard font into RAM, read the DATA statements, incorporate the new character designs and set the default window to use the new font.

● *In next month's SuperBasic more character manipulation will be explained.*

```
Listing "50"

5000 DEFine PROCedure WriteIn
5005 LOCal X, loop, D$, Cbase, CharBase
5010 Base = RESPR(900): Cbase = 44442
5015 FOR X = 0 TO 866 STEP 4
5020   POKE_L Base+X, PEEK_L(Cbase+X)
5025 END FOR X
5030 PRINT#0; "QL set copied.": RESTORE 5102
5035 REPeat loop
5040   READ D$: IF LEN(D$) > 2: EXIT loop
5045   CharBase = Base -277 +CODE(D$) *9
5050   FOR Cbase = CharBase TO CharBase +8
5055     READ Byte: POKE Cbase, Byte
5060   END FOR Cbase
5065 END REPeat loop
5068 Cbase = PEEK_L(PEEK_L(163960) +4)
5069 POKE_L Cbase+42, Base
5070 PRINT#0; "Font installed"
5075 END DEFine WriteIn
```

DIY TOOLKIT

Simon Goodwin adds new commands to QL SuperBasic which let you keep several SuperBasic programs in memory and swap between them instantly.

QL SuperBasic is a great programming language but it has a strange limitation for a multitasking machine. You can have only one Basic program in memory at a time. If you want to edit or run a different program you must throw away the existing program and load another in its place. MultiBasic lets you keep several Basic programs in memory at one time and swap between them at will. You can edit or run the current program in the usual way, while the others are held as separate QL tasks.

This column introduces four SuperBasic commands – UNLOAD, RELOAD, RESAVE and REMOVE. They have been tested successfully on the QL, Minerva and the Thor XVI. They occupy less than 500 bytes of memory when loaded.

You can type-in the listings and use the commands in your own programming. There is insufficient room to note all the ramifications this month but my next column will explain how the commands work and explore some of their implications.

Easy

Interactive testing is easy with MultiBasic because it saves the complete programming environment, including variable values, arrays and temporary results. You can UNLOAD a program at any time, even while it is running, and when you RELOAD it, it will continue where it left off.

MultiBasic keeps track of all the local variables and procedure calls in progress. You can break into a program, UNLOAD it and CONTINUE later. In the meantime you can load or RELOAD any number of other Basic programs. This is very useful when working with Basic compilers, which expect you to load programs into the interpreter before you compile them. It means you can use small utilities, written in Basic, without losing the big program on which you may be working.

The bigger your programs the more time you can save with MultiBasic. SuperBasic loading becomes slower and slower as program size increases, while MultiBasic RELOAD time is proportional to the length of your program and less than a second even for programs with thousands of lines. Once you have loaded a program you never need re-load it the slow way until you run out of memory or turn off the computer.

MultiBasic is specially useful because Basic loading is so slow on the QL. The Microdrive light flicks on and off, tugging the tape in short bursts, as the QL reads the program text in 512-byte portions and converts it into internal 'tokens'. The conversion takes a long time, whatever the device. Disc loading is almost as slow as Microdrive; even RAM discs are painfully slow, because the text must be

```
* Version 3.2   Copyright 1988-90   Simon N Goodwin
*
looper  equ     24830           Opcode of BRA.S *-2
*
start   lea.l   define,a1
        move.w  $110,a2         BP.INIT vector
        jmp     (a2)            Add new commands
*
*       SHARED KEYWORD CODE
*
unload  moveq   #-1,d7          Flag for UNLOAD
        bra.s   getname
reload  moveq   #1,d7           Flag for RELOAD
        bra.s   getname
resave  moveq   #-2,d7          Flag for RESAVE
        bra.s   getname
remove  moveq   #0,d7           Flag for REMOVE
*
getname lea     8(a3),a4        Check for one parameter
        cmp.l   a4,a5
        beq.s   one_par
bad_par moveq   #-15,d0         Report BAD PARAMETER
dropout rts
one_par tst.b   0(a3,a6.l)      Does it have a value?
        beq.s   unset           No, use the name instead
        move.w  $116,a0         CA.GTSTR vector
        jsr     (a0)            Get the value as a string
        bne.s   dropout         Give up if that went wrong
        move.l  $58(a6),a1      Get string offset (BV.RIP)
        tst.b   0(a1,a6.l)      Check length is 0-255
        bne.s   bad_par         No good, length >= 256
        lea.l   1(a1),a5        0(A5,A6.L) -> length byte
        bra.s   tst_len
*
unset   move.l  24(a6),a0       Find BASIC's Name Table
        move.w  2(a3,a6.l),d0   Index of actual parameter
        lsl.w   #3,d0           N.T. entries take 8 bytes
        adda.w  d0,a0           (A0,A6.L) -> N.T. Entry
        move.w  2(a0,a6.l),d0   D0 is name offset in list
        move.l  32(a6),a5       Find BASIC's Name List
        add.w   d0,a5           0(A5,A6.L) -> length byte
tst_len tst.b   0(a6,a5)        Check name is 1-127 chars
        ble.s   bad_par
*
* Look up the required name amongst all the tasks
*
look_up moveq   #0,d1           Start with task 0,0
        bra.s   scan            Scan all the tasks
look_on tst.l   d1              At end of task tree?
        beq.s   do_cmd          If so, do the command
scan    moveq   #0,d2           Scan from top of tree
        move.l  d1,d4           Save current task ID
        moveq   #2,d0           MT.JINF trap key
        trap    #1              Find the next task
        tst.l   d0              Does it exist?
        bne.s   do_cmd          If not, do the command
*
* A0 -> task, D4 is task ID, D1 is ID of 'next' task
*
        move.l  a0,d6           Save base address of task
        addq.l  #2,a0           Skip BRA.S at the start
        move.l  (a0)+,d5        Get length
        cmpi.w  #$4AFB,(a0)+    Does task have a header?
        bne.s   look_on         No, don't look for a name
        tst.b   (a0)+           Name length should be <256
        bne.s   look_on         Otherwise, keep looking
        move.b  0(a5,a6.l),d0   Get parameter name length
        cmp.b   (a0)+,d0        Do name lengths match?
        bne.s   look_on         No, try another task
        move.l  a5,a4           Copy name pointer for test
compare move.b  1(a4,a6.l),d2   Get parameter character
        moveq   #%11011111,d3   Mask to ignore letter case
        and.b   d3,d2           Convert parameter to caps
        and.b   (a0)+,d3        Convert header to capitals
        cmp.b   d2,d3           Compare characters
        bne.s   look_on         Look on unless they match
        addq.l  #1,a4           Advance through parameter
        subq.b  #1,d0           Count one less to check
        bne.s   compare         Check all the characters
        moveq   #-1,d1          Flag that name was found
*
* If D1 is negative, task was found: code at D6, length D5
*
do_cmd  tst.l   d7              What operation is needed?
        bmi.s   unloda          Negative, UNLOAD / RESAVE
        bne     reloda          Positive, RELOAD
*
* REMOVE routine - if we found a task, we can remove it
*
remova  tst.l   d1              Was the name found?
        bmi.s   kill_it
bad_job moveq   #-2,d0          Report INVALID JOB
        rts
exists  moveq   #-8,d0          Report ALREADY EXISTS
        rts
unknown moveq   #-7,d0          Report NOT FOUND
        rts
*
kill_it move.l  d4,d1           Set up task ID
        moveq   #5,d0           MT.FRJOB trap key
        trap    #1              Remove the task
        rts                     Report back to caller
*
* UNLOAD routine - copies BASIC to an image in task RAM
*
unloda  tst.l   d1              Did we find the name?
        bpl.s   make_it         No, so create it
        addq.w  #1,d7           Is this UNLOAD?
```

converted into tokens and that takes much longer than fetching the raw data.

MultiBasic stores the program in its tokenised form, so it can be moved through memory directly into the interpreter. MultiBasic can move 270K per second, even on the slowest QL, so it is much faster than existing 'quick loaders'. It can load about one megabyte per second on 16-bit machines like the Thor or Atari QL emulator and manages 500K a second on a QL with fast memory.

MultiBasic makes it easier to write often-used utility programs in Basic, as there is no need to re-load them – and tokenise them – whenever they are needed. It is particularly useful for 'template' programs which need regular tweaks for specific tasks.

MultiBasic also speeds SuperBasic program development and testing. You can UNLOAD a copy of a program before making changes and then RELOAD it instantly if problems occur in the revised version. It is still a good idea to SAVE to disc or cartridge every so often, in case you lose the entire contents of memory. Thankfully, SAVE is much faster than LOAD.

When you UNLOAD or RESAVE a MultiBasic task the current command line is saved along with the program. Any commands after the UNLOAD will be executed as soon as you RELOAD that image, so:

```
UNLOAD prog : LIST
```

saves 'prog' in such a way that it is LISTed automatically whenever you RELOAD it later. Alternatively, you can make a program RUN as soon as it is RELOADed by saving it like this:

```
RESAVE prog : RUN
```

Extra statements after RELOAD are ignored; like LOAD and LRUN, RELOAD discards the remainder of the command-line when it loads a new program. It is acceptable to use UNLOAD inside a program as it runs. When the program is RELOADed it will continue running from the point after the UNLOAD statement, as if it had never stopped. You can arrange sequences of programs which load one another and communicate through the MEM device featured in DIY Toolkit last summer.

MultiBasic reports 'out of memory' if there is insufficient free memory to UNLOAD or RESAVE a task. It complains 'bad parameter' if you do not specify a valid task name. It accepts any sequence of 1-127 characters, in single or double quotes, or a string variable or expression.

Unquoted names are allowed so long as they are not used for some other purpose in the program. The file commands SAVE, LOAD, LBYTES and so on use the same rules, although they do not

allow such long names. The shortest practical names are single letters or digits.

The 'invalid job' report means that you have tried to RELOAD or REMOVE a task which does not exist or there are so many tasks that Qdos has insufficient space to keep track of another one, which is very unlikely.

You may get a 'channel not open' error if you RELOAD a task which was using channel numbers above #2 when you UNLOADed it. UNLOAD saves a copy of the current task but it leaves the original in SuperBasic.

When you type CLEAR, CLOSE, LOAD, LRUN or NEW, the interpreter

```
        beq.s   exists          If so, it already exists
        bsr.s   kill_it         Otherwise it's RESAVE...
make_it moveq   #0,d0           MT.INF trap key
        trap    #1              Find system variables
        trap    #0              Stop anything moving!
        or.w    #$700,sr        Disable interrupts
        move.l  a0,a3           A3 -> system variables
        move.l  20(a0),d3       D3 -> End of BASIC+1
        sub.l   16(a0),d3       D3 = Length of BASIC area
*
* D3 is length of 'data' area. Get 'code' length into D2.
*
        moveq   #18,d2          D2 = Minimum 'code' header
        move.b  0(a5,a6.l),d5   Find length of task name
        addq.b  #1,d5           ... and ensure it is even
        bclr    #0,d5           Round length to even
        add.b   d5,d2           Add name to 'code' length
*
* Allocate transient program memory for the image
*
alloc8  moveq   #0,d1           Owner: SuperBASIC
        move.l  d1,a1           Put task in TRNSP
        moveq   #1,d0           MT.CJOB trap key
        trap    #1              Create a transient task
        tst.l   d0              Check for errors
        bne.s   unlock          Give up if CJOB failed
*
* Set up a dummy task header
*
        move.w  #looper,(a0)+   START is BRA.S START !
        move.l  d3,(a0)+        Store length AFTER header
        move.w  #$4AFB,(a0)+
        add.l   a6,a5           A6 cannot change now
        move.b  (a5)+,d0        Get length of task name
        ext.w   d0
        move.w  d0,(a0)+        Store length (always >0)
setname move.b  (a5)+,(a0)+     Copy name a byte at a time
        subq.b  #1,d5           Use even count in D5 to
        bne.s   setname         ... ensure A0 ends up even
        move.l  a6,(a0)+        Save base address of BASIC
        move.l  usp,a5          Save User Stack Pointer
        move.l  a5,(a0)+
*
        move.l  16(a3),a3       A3 -> NEW start of BASIC
copy_d3 lsr.l   #3,d3           Count pairs of long words
        subq.l  #1,d3           Adjust for DBRA loop
copier  move.l  (a3)+,(a0)+     It's fast and safe to move
        move.l  (a3)+,(a0)+     ... eight bytes at a time
        dbra    d3,copier       Move 270-540 QL K a second
lockout moveq   #0,d0           Report no error
unlock  andi.w  #$D8FF,sr       Re-start QDOS
        rts
*
* RELOAD <name>  - fetch task from (D6), length in D5
*
reloda  tst.l   d1              Did we find the name?
        bpl.s   unknown
load_it move.b  0(a5,a6.l),d7   Find name length for
                                later
        addq.b  #1,d7
        bclr    #0,d7           Round length to even
        moveq   #0,d0           MT.INF Trap key
        trap    #1              Find system variables
        move.l  a0,a4           Save system var. pointer
        move.l  20(a0),d1       D1 -> End of BASIC+1
        sub.l   16(a0),d1       D1 := Length of BASIC area
        sub.l   d5,d1           D1 is delta size
        beq.s   size_ok         No change required
        bpl.s   shrink          Discard excess space
        neg.l   d1              Expand by D5-D1 bytes
        moveq   #22,d0          MT.ALBAS trap key
        trap    #1              Expand BASIC work space
        tst.l   d0              Check for OUT OF MEMORY
        beq.s   size_ok
        rts
shrink  moveq   #23,d0          MT.REBAS trap key
        trap    #1              Shrink BASIC work space
*
* Now work out the address of the code in the task space
*
size_ok moveq   #10,d2          Skip fixed part of header
        add.b   d7,d2           Skip name (max length 127)
        add.l   d2,d6           Point D6 at saved A6 & A7
        move.l  d6,a3           Set source pointer
        trap    #0              Stop BASIC space moving
        or.w    #$700,sr        Disable interrupts
*
        move.l  a6,a0           Offset user A7 by delta A6
        sub.l   (a3)+,a0        Delta address of BASIC
        add.l   (a3)+,a0
        move.l  a0,usp          USP is delta A6 + old USP
        move.l  16(a4),a0       Get destination, SV.BASIC
        lsr.l   #3,d5           Count long word pairs
        subq.l  #1,d5           Adjust for DBRA
copy    move.l  (a3)+,(a0)+     Move task data to BASIC
        move.l  (a3)+,(a0)+
        dbra    d5,copy
        bra.s   lockout         Restart QDOS and exit
*
define  dc.w    4               Number of procedures
        dc.w    unload-*
        dc.b    6,'UNLOAD'
        dc.w    reload-*
        dc.b    6,'RELOAD'
        dc.w    resave-*
        dc.b    6,'RESAVE'
        dc.w    remove-*
        dc.b    6,'REMOVE'
        dc.w    0,0,0           No functions
        end
```

closes all channels except the standard windows #0, #1 and #2. If you use those commands, RELOAD a program and CONTINUE from part-way through, Qdos will give an error if you try to use other channels without re-opening them first.

MultiBasic stores the names and addresses of current resident extensions such as DIY Toolkit routines when you UNLOAD. If you load extensions part-way through a session they will not be available while you RELOAD earlier files. Absent procedures cause a 'bad name' error, while missing functions give 'error in expression'.

MultiBasic tasks can be treated like other tasks. They share the same format and area of memory with other tasks, so you can manipulate them with standard Toolkit commands. There is no need for a new command to list the MultiBasic tasks – you can use common instructions like JOBS or LIST_TASKS. You can also dispose of MultiBasic tasks once they are no longer useful with RJOB, REMOVE _TASK or the DIY Toolkit PURGE command.

MultiBasic includes its own REMOVE command which will dispose of any task, given its name in upper- or lower-case. REMOVE can be used on machine code tasks as well as SuperBasic. You can stop a clock with:

REMOVE CLOCK

The only limitation is that you must know the name of the task you want to remove. Some tasks, like the Psion Quill, do not have a name, so they cannot be

clobbered by REMOVE. You must use RJOB or some other command which lets you specify a task by number.

Limitation

MultiBasic will not create two tasks with the same name. If the name you UNLOAD is in use, MultiBasic reports 'already exists'. You must pick either another name, or RESAVE, which will over-write the old task. If you load the same machine code task several times, REMOVE will get rid of the first one with a matching name every time you use it. REMOVE scans tasks in the same order as JOBS and LIST_TASKS.

The MultiBasic code is listed in two forms. Listing one is the commented assembly code which will be explained in my next column. You will need an assembler to convert this into code you can run; I use HiSoft *Devpac*. Listing two is a simple Basic loader which reads the machine code from data statements and stores it in a file. It is easier to enter this listing, especially as lines 100-580 are the same for every DIY Toolkit package but it is relatively difficult to customise the data. Once you have created the code file you can add the new commands to SuperBasic, like this:

```
base=RESPR(446)
LBYTES "filename",base
CALL base
```

Thereafter you can use UNLOAD, RELOAD, RESAVE and REMOVE in your own programming.

In my next column I shall delve into listing one and explain how MultiBasic works. The code includes several new tricks for DIY Toolkit readers, including task handling and case-independent comparison. It also shows how you can pick up 'name' parameters without the need for the user to type quotation marks.

I have been developing MultiBasic since the summer of 1988 and the version listed has been condensed for the magazine, so I plan to explain a few ways the code can be expanded and improved, along with notes on the implications of the new commands.

● *After more than three years I am running out of ideas for new QL commands and functions which will fit the DIY Toolkit format. Please send your suggestions if you would like me to explore a specific areas in this column or to implement new and original commands.*

```
100 REMark Sinclair QL World HEX LOADER
110 REMark by Marcus Jeffery & Simon N Goodwin
120 :
150 CLS: RESTORE : READ space: start=RESPR(space)
160 PRINT "Loading Hex..." : HEX_LOAD start
170 INPUT "Save to file..."$f$
180 SBYTES f$,start,byte : STOP
190 :
200 DEFine FuNction DECIMAL(x)
210 RETurn CODE(h$(x))-48-7*(h$(x)>"9")
220 END DEFine DECIMAL
230 :
240 DEFine PROCedure HEX_LOAD(start)
290 byte = 0 : checksum = 0
300 REPeat load_hex_digits
310    READ h$
320    IF h$="*" : EXIT load_hex_digits
330    IF LEN(h$) MOD 2
340       PRINT"Odd number of hex digits in: "$h$
350       STOP
360    END IF
370    FOR b = 1 TO LEN(h$) STEP 2
380       hb = DECIMAL(b) : lb = DECIMAL(b+1)
390       IF hb<0 OR hb>15 OR lb<0 OR lb>15
400          PRINT"Illegal hex digit in: "$h$ : STOP
420       END IF
430       POKE start+byte,16*hb+lb
440       checksum = checksum + 16*hb + lb
450       byte = byte + 1
460    END FOR b
470 END REPeat load_hex_digits
480 READ check
490 IF check <> checksum
500    PRINT"Checksum incorrect. Recheck data.":STOP
520 END IF
530 PRINT"Checksum correct, data entered at: "$start
560 END DEFine HEX_LOAD
570 :
580 REMark Space requirements for the machine code
590 DATA 446
600 :
610 REMark Machine code data
620 DATA '43FA018C34790000','01104ED27EFF600A'
630 DATA '7E0160067EFE6002','7E0049EB000BBBCC'
640 DATA '670470F14E754A33','E800671A30790000'
650 DATA '01164E9066EE226E','00584A31E80066E2'
660 DATA '4BE900016016206E','001B3033E802E748'
670 DATA 'D0C03030E8022A6E','0020DAC04A36D000'
680 DATA '6FC0720060044A81','673C7400280170D2'
690 DATA '4E414A8066302C08','54882A1B0C584AFB'
700 DATA '66E44A1866E01035','E800B01B66D8284D'
710 DATA '1434E80176DFC403','C618B60266C852BC'
720 DATA '530066EC72FF4A87','6B1C6600008B4A81'
730 DATA '6B0C70FE4E7570F8','4E7570F94E752204'
740 DATA '70054E414E754A81','6A06524767E861EE'
750 DATA '70004E414E40007C','0700264826280014'
760 DATA '96A8001074121A35','E800520508B50000'
770 DATA 'D405720022417001','4E414A8663030FC'
780 DATA '60FE20C330FC4AFB','DBCE101D48B030C0'
790 DATA '10DD530566FA20CE','4E6D20CD266B0010'
800 DATA 'E68B538320DB20DB','51CBFFFA7000027C'
810 DATA 'DBFF4E754A816AB2','1E35E80052070887'
820 DATA '000070004E41284B','222B001492AB0010'
830 DATA '928567126A0C4481','70164E414AB06706'
840 DATA '4E7570174E41740A','D407DCB226464E40'
850 DATA '007C0700204E91DB','D1DB4E60206C0010'
860 DATA 'E68D538520DB20DB','51CDFFFA609E0004'
870 DATA 'FE7C06554E4C4F41','4400FE760652454C'
880 DATA '4F414400FE700652','455341564500FE6A'
890 DATA '0652454D4F564500','000000000000','*',40312
```

FLASHBACK 2

Bryan Davies conducts a comparison between Flashback 1 and Flashback 2 for those thinking of upgrading.

There are two versions of *Flash-Back* available and this article is based using the enhanced Special Edition version, otherwise known as FlashBack 2. The bulk of the functions provided are common to both versions. As the cost of an upgrade from FlashBack 1 to 2 is low and the extra functions are desirable, it is suggested that the change is well worth making; but consider how much spare memory your QL has before making the change.

One significant difference between the two versions is the memory space required; it is appreciably greater for the SE version. Another difference is the facility to have more than one copy of FlashBack running concurrently. The two factors may be related for some users. In my system, the original FlashBack provided an answer to a problem which could not be solved with Archive – that of running two unconnected databases simultaneously. It was possible to combine the two into one file, while still being able to treat them separately when necessary.

Split files

Unfortunately, the SE version caused the QL to run short of memory space, with the result that the combined file had to be split once more. As it is possible to run two copies of FlashBack SE without a great memory overhead, the practical effect is that access to any of the data in either file is as fast as ever; the total memory used is about 20-30KB more and I can now choose not to have the larger of the two databases active unless it is really required.

Suppose you have a name and address database, with telephone numbers, which needs to be active all the time. My _DBA file of 432 records takes about 50KB of disc space. Then consider a less-often-used database, which you can afford to load only when needed and kill afterwards if space is required. My example is a German-English dictionary of 2,073 records which takes about 127KB of disc space. Back-tracking to Archive days, these two together took about 40KB more space as _DBF files.

Because even run-time Archive needs about 50KB in which to work, running each database under its own copy of Archive required close to 300KB. In addition, because of the nature of the Psion programs, you have to use a switching program as well and that will take at least another 10KB, possibly much more but you can now make Archive EXEC-able and switch with CTRL+C as explained in the DIY Tookit article in the October, 1989 issue.

Saving memory

It is clear that moving the databases into FlashBack can save memory but the advantage is more than that. I ran one copy of Archive and had a swapping routine to change databases but it took several minutes to make the change from one to the other and that is not much use when you need to look up a telephone number or word in a hurry. The equivalent _DBA files load and save much faster. Since FlashBack is a pop-up program, your two copies can be run alongside one or more other programs, without the need of a special switching program.

The only area in which the user can exercise control over memory usage is the default setting for Work Space, Save Screens, References per Record, and Maximum Number of New Records to be added per session, which can be altered with the CONFIG_BAS program; in this program, Work Space is referred to as Maximum Text Expansion.

When buying FlashBack 2 do not tamper with any of those settings until the instruction files have been loaded and re-saved, because those compressed files require higher values for work space, references per record and so on than you may need in daily operations. The Work Space figure shown at the left end of the status area is a direct measure of the number of characters which can be added before the program says enough. Generally, this should be the multiple of the number of records likely to be added in one session and the average record size.

If records are typically two full lines long – about 160 characters – and you do not expect to add more than 30 of them per session, the Work Space required is 160 x 30 = 4,800. Bear in mind that it is a quick process to save the database and reload it and that will restore the work space to its set maximum, so it is no disaster to have it reduce to zero now and then. Many

databases will stabilise after a fairly short period of use, with few records being added; I rarely need more than 1,000 characters work space and a corresponding maximum New Records figure of 20. The References per Record figure is not well explained in the instructions and I do not pretend to understand fully its significance; suffice it to say that if you do not use sub-records a value of 1 is likely to be sufficient.

A memory area not covered by the CONFIG_BAS program is that required to store the FlashBack screen while it is not being displayed. If you elect to answer N to the Save Screen option it is of no consequence what size the FB screen is; it is not saved and has to be re-drawn each time FB is called up, so there is no memory tied-up in holding its details. This is more an irritation than a real hardship, as the programs over which FB is to be called up will usually have their own screen-refresh keying – e.g., F4 for *The Editor* and *text87*, SHIFT+F5 for Psion programs, – which can be invoked to get rid of the FB display.

On the other hand, if you answer Y to this option you are causing up to 32KB to be reserved for the FB display details. The smaller the display the less money is required to save it; a full-width, minimum-height display takes 13KB and a minimum-size display takes 5KB; the memory taken is noted on the re-sizing screen, accessed by keying SHIFT+F4. Pruning all membry usage can save about 40KB, which may make all the difference between getting FB in comfortably with your usual applications programs and having to remove some programe you really need available all the time.

You may suffer the same delusion I do; if the window used by FB is small you tend to regard the program as being only a small thing. That is an injustice to the program and it can leave you wondering where all the memory has gone, forgetting that FB may well be taking several hundred KB of it.

Searches are accomplished remarkably fast, the speed seeming to be the same regardless of the size of the database and the position(s) of instances of the specified search string in it. As with other operations you need to check what field is current on the status line before starting a Search. You can become very irritated by seemingly-inconsistent behaviour, induced by your failure to set the appropriate field in which searching occurs. When a global search is required – looking for a string of characters anywhere in any record – the Field 00 All setting is necessary.

Searching for a string in a specific field requires that field to be the current one at the left of the status line when the Search is initiated. FB will not search for a string in a specific field – it looks only at the start of a field. The search is insensitive to letter case, so that upper or lower can be typed-in.

One of the added functions which take FB to a higher level of capability is the facility to specify records which are to be included in, or excluded from, Group-ing. The Include and Exclude commands enable the user to tailor a Group, to circumvent the relative inflexibility of the Group command. You may find at some stage that you want to delete a number of records from a database and there are too many of them to be removed quickly using the Delete command on each of them individually.

Look for some common factor in these records; perhaps you now have enough Biggles books to take them out of the main book list and give then a separate listing and they can be identified by the one word "Biggles". The Group command can then be used, with "Biggles" inserted on the status line as the Group string; this will work only if you have a field – e.g., Category – which starts consistently with "Biggles", because the Group command does not look within a field, only at the start of it.

Once the Group has been created you may find some of the records in it do not really concern Biggles books; perhaps there is an entry "Biggles look-alike" in the Category field, indicating that an author has copied Captain Johns' style. Such records can be marked by making them current and using CTRL+X to exclude them.

Grouping

When the keying is used the current record apparently disappears, although it has merely been put into the background with all the other records which did not meet the Group specification. There may also be some records which are not of Biggles books as such but which you feel should be included with them – reference works on Johns' writings, perhaps. Mark them the same way but using CTRL+I. I hope it is obvious that you have to mark records for inclusion before Grouping – they will not be available for inclusion thereafter. Note that the Include and Exclude functions are not mutually exclusive; they can be used together when creating any Group. The total number of records now current is noted on the status line, against "Ref No".

The previous paragraph began as a note on deleting records but apparently ended with those records being the only ones left in the database. Remember that the fact that records are not currently visible, or accessible, does not mean that they have been removed from the database. Both the Group-ed and non-Grouped records are still there but you now have the possibility of handling them differently. Use the CTRL+W command to Write the Group out to disc. You must use a different _DBA file name from the one for the current (complete) database or you will over-write the latter. The newly-saved file can then re reloaded later, as a Biggles-only database. Returning to consideration of your main database, the process of Group-ing on Biggles did not reduce the number of records.

To delete the Biggles records from the main databse, Kill the Group to get all records displayed again, then Index on the Field containing the Biggles reference; if that word is the first in the field it should put all the unwanted records next to each other. Go to the first record to be deleted and key CTRL+X to exclude it. It should be possible to go through the whole batch of unwanted records quickly by holding down the CTRL+X keys until all those records have been excluded. This is safer than using Delete, because you can always make the excluded records current again by keying CTRL+K if you have excluded a record in error. It is also a faster operation, with less keying.

Once all unwanted records have been excluded from the current display you can Write out the file and then re-load it, when it will contain only the records which do not concern Biggles books. Always take care when Write-ing the database to disc if you have been performing such separation operations; check that the file name on the status line is the one you want.

A big improvement of FB2 over FB1 is the facility to make successive Group operations to get at sub-sets of records. FB1 would allow you to Group on, say, all records having a field starting with "Tel. (01)", to reduce the current database to records including London telephone numbers but you could go no further than that. FB2 allows you to Group as many times as required. The next step might be to Group on "Billabong", to reduce the current database to records of people, with London area telephone numbers, having that surname – first name, perhaps?

If that does not reduce you to zero records – impossible, because all records are brought back if you try to delete the last one in a Group – you could have a go at the address, by Group-ing on "East Cheam", for instance. Having said all that, reducing the number of current records makes no perceptible difference to the speed with which FB operates. A string search will be just as fast for 1,000 current records as for 10. The advantage is when using other commands – Write or Print.

The ALTKEY and Last Line Recall functions of Toolkit II are very useful, and can be very irritating, too. If often takes a good deal of experiment to get the desired result from an ALTKEY setting and you may make frequent use of Last Line Recall to modify the definiton. The recalled line is regularly incomplete or incorrect, even with later Toolkit versions; this is particularly true with longer lines.

So to circumvent this try typing the ALTKEY definition into a FB record. You can use the CTRL+T command to Transfer the definition to the SuperBasic line, so that you know the recall process will be correct each time. An FB record can be

used in place of an ALTKEY text setting. Standard positions of text of almost unlimited length can be deposited into an underlying program quickly.

Increasing database file sizes leads to lost time looking for particular records. Finding one out of 10,000 records when you are not sure of the contents can be frustrating. The View command – either as a result of a Group operation or on its own – enables you to cursor through records quickly but that is still a fairly slow process when there are many records. FB2 enables you to initiate a string search from the View window.The search is made from the start of the field shown on the View screen and it is necessary to make the appropriate field current by the Group command, if it is not already there.

Case sensitive

After the word "View" on the status line, a cursor blinks awaiting input of a search string. Typing-in the first letter of a word for which you are looking makes the first record which has the letter as the first one in the field the current one. The highlight in the View screen may be on a record with "Adams" as the first word in the highlighted field and typing "K" on to the status line will then take the highlight to a record with, say, "Jones" as the first word in that field. Adding a "u" after the "J" might then take the highlight to "Justice". and so on.

Note that the search in case-sensitive and the first letter of the string is likely to be a capital. When such a search yields no action check that you are giving the string letters the correct case and that the file is Group-ed or Index-ed on the field to be searched. If no Group or Index command has been given the search will be made on Field 01. In the case of my name and address file there is no Field 01 in use and a search from the View window yields nothing until the Group or Index command has been used with the Field set to 03 or 04.

It may be worth pointing out that you are not obliged to start records with Field 01. In the case of my file it is now a historical matter that the first field is 03 but it could have future use. You can have more than one database contained in one file if you settle on a good way of separating them when necessary. Making one database from Fields 01 and 02 and another from Fields 03 and 04 gives you a simple way of separating them; Group on one field and you will exclude all records which do not use that field.

My original file had this structure until FB2 appeared on the scene; the greater size compared to FB 1 compelled me to split the file into two parts,which was done easily by Group-ing first on Field 01 and Write-ing the file out, then Kill-ing the Group and creating another, based on Field 03, and Write-ing that out to another file. It would be easy at some time in the future to re-combine the two databases, if memory space permits use of the larger file

When deciding on the layout of fields in a database file, bear in mind the form(s) the output will take. If queries are to be shown on the screen only, fields can be packed as closely as makes sense visually. Where a printout is required the field layout should take account of that layout. The obvious case is a name and address file. from which lists or labels may have to be printed. The various parts of the name and address should be in separate fields. It is not necessary for the fields to be on separate lines, since the lines on which they are to be printed can be specified when using the Report Generator function.

If printing is to be done direct from FB, however, the separate parts of the name and address should be on separate lines in the records. The Field Name is not printed; thus, for example, "TEL." needs to be in the field contents as well as being the field name if it has to be printed. Bear in mind that the Print command will act on the whole file if you do not specify a Group of records; printer paper may soon be rushing out all over the place.

A big advantage over Archive is the ability to add fields easily and quickly. The first step, as always, is to save and back-up the current database. Key CTRL+E to Edit field contents. If you want to insert a new field between the current fields 05 and 06, use the cursor to make Field 06 current, then key F5 and type-in the name alongside the 05. You have a choice of two actions at this point – either key ESC, in which case you will be returned to the current record with nothing changed other than that what was previously Field 06 will now be 07, or key ENTER, which will cause a marker to be inserted for the new Field 06 at the cursor point.

It is easy to consider FB as a database in the 'classical' sense – a series of similar records. This is to miss a good deal of its usefulness, as it is a very flexible program. Perhaps you need to make a note, or even create a complete document, but all the memory space is taken by a spreadsheet and program, plus FB.Create a new record in FB and type the text, free-form, preferably starting at a field number different from those used by the main database records; that makes it easier to find subsequently.

When you load your WP program later the text can be Transfer-red to it and the record deleted. A record can be used as a development tool for ALTKEYs, for SuperBasic programs, or for almost anything. Most people need to make notes at some time and FB can always be popped-up to take them. Perhaps you should keep your "Jobs To Do" list this way. Although there is no cut and paste facility in FB the paste part is effectively there; the same record or field can be put into several programs using the Transfer command.

The Archive Order command is a very necessary tool; it puts records in a sensible sequence and it permits use of the Locate command, which finds strings much faster than Find. The FB Index command is similar but some differences should be borne in mind. You cannot back-up an Index-ed file to reduce the file size as you can do with Order-ed Archive Files. You will not speed searches by Index-ing files; it may slow searches but it is unlikely you will be aware of this.

One thing Index-ing does not do is re-order the individual records in the disc file, because FB works only on the memory file. Once the Index command has been given you can Write the file out and it will remain in the Index-ed order when next Read in; the order will revert to that in which records were typed-in once the Kill command is used. So long as access is fast it should be of no concern to the user what order the records are in on disc.

There is a choice of three forms for the Index command, with two more choices associated with those three. The default form, as displayed on the status line when CTRL+I is first keyed, is 2 ABCabc. This is the form used by FB1. Text beginning with capital letters will be sequenced before the beginning with the lower-case – e.g., AA Aa B X aaB ab. That applies to each character in the string and an Index string can be up to 500 characters long, stretching over several fields. This means that you are not limited to sequencing records ona limited amount of data, such as surnames in a name and address file.

Changing the form

Pressing a Space Bar before the Index-ing process is started will change the form to one of the two alternatives. 3 aBcDEf places records in the order they occur – there is no attempt to place capitals before lower-case or vice versa. The previous examples could be in the order a Aa AA ab aB B X or a AA Aa aB ab B X, or with other variations to the positions of the first five strings. The sequence will always take account of whether a letter is before or after another in the alphabet. This might be called the "dictionary method" of sorting. Another press of the Space Bar before the Index gives the third choice – 1 AaBbCc. This keeps letters in alphabetical order but puts capitals before lower-case for each letter encountered.

The final two choices concern numbers. FB1 was rigid in the way it looked at numeric characters; it figures that the key-code for 1 comes before that for 9 – see "character set and keys" in the Sinclair QL User guide – therefore 10 must be placed before 9 in a sequence. You could circumvent this by placing a 0 before numbers like the 9 in this example but that was something of a nuisance. FB2 will still sort in this way by default but you can key N when Index is on the status line and make it use the more sensible method of putting 9 before 10. The status line then looks for this form – N3 aBcDEf – or either of the other forms, preceded by N.

THE P·R·O·G·S

If you have a program worthy of consideration, send it to 'The Progs', Sinclair QL World, Greencoat House, Francis Street, London SW1P 1DG. We pay for everything published at the usual page rates.

REVERSE POLISH CALCULATOR

by Peter Laurisden

RPN_CALCULATOR_BAS is a program which emulates a Reverse Polish Notation Scientific Calculator. RPN is a system of representing mathematical equations. Fewer keystrokes are required to do complex calculations with a RPN system than with the regular system. In the RPN system there are no parentheses and no = (equals) keys. Only two numbers are worked with at a time.

An abreviation of the functions/commands available is displayed at the top of the screen. The left-hand part of the screen is used by the command-line, the registers and the error messages. The right-hand side is used by the memory, the converter and the statistical function. The bottom of the screen is used for interactive purposes and for displaying HELP messages.

The following are the Help texts available from the program:

If two numbers are entered and "+" is pressed Z2 and Z1 will be added.

If a new number is entered and "+" is pressed, Z2 and Z1 will be added.

If "+" is pressed again, the result in Z3 and Z1 will be added.

After a conversion the calculator will work on Z2 and Z1.

* + / − are used to multiply, add, divide and subtract Z1 and Z2 (or Z3).

" ": Raises the number in Z2 (or Z3) to the power of the value in Z1.

"INV": Calculates the inverse value (1/x) of the number in Z1. 1/Z1➔Z1.

"NEG": Negate. Multiply the number in Z1 with −1. Z1➔Z1*(−1).

"RO": Round off the value in Z1 to a chosen number of digits, e.g. 2.345➔"RO" 2 <ENTER>➔2.35.

"CLR": Clears all Z-registers of their content. Warning: Values not stored in Memories are lost.

"SIN": Compute the sine of Z1 which is taken to be in degrees.

"ASIN": Compute the arcsine of Z1 in degrees.

"COS": Compute the cosine of Z1 which is taken s degrees.

"ACOS": Compute the arccosine of Z1 in degrees.

"TAN": Compute the tangent of Z1 which is taken to be in degrees.

"ATAN": Compute the arctangent of Z1. Result in degrees.

"SEC": Compute the secant of Z1 which is taken to be in degrees.

"ASEC": Compute the arcsecant of Z1. Result in degrees.

"COT": Compute the cotangent of Z1 which is taken to be in degrees.

"ACOT": Compute the arccotangent of Z1. Result in degrees.

"CSC": Compute the cosecant of Z1 which is taken to be in degrees.

"ACSC": Compute the arccosecant of Z1. Result in degrees.

"PI": Put the value of π in Z1. = 3.141593.

"LN": Complete the Natural Logarithm of Z1. Base e=2.7182818.

"E": Compute the exponential of Z1. (e x).

"LOG": Compute the logarithm of Z1.

"%": Multiply Z1 with 0.01.

"N!": Compute the factorial of Z1, e.g. 4! = 4*3*2*1 = 24.

"STAT": Sub-program. Compute the statistical data of a set to observations. You will have to write down the results, as they are lost when you return to the calculator.

"HYP": Compute the value for the hypotenuse of a triangle, given two sides.

"SIDE": Compute the value for a side of a triangle given the hypotenuse and other side. Z1(side1), Z2(hyp)➔Z1(side2), Z2(side1), Z3(hyp).

"DEG": Change from radians to degrees. Z1(rad)➔Z1(deg).

"RAD": Change from degrees to radians.

"POLR": Compute polar co-ordinates given rectangular X=Z2 Y=Z1 R=Z2 ∅=Z1.

"RECT": Compute rectangular coordinates given polar R=Z2 ∅=Z1 X =Z2 Y=Z1.

"MET": Compute metric conversions.

"HSIN": Compute the hyperbolic sine of Z1.

"HCOS": Compute the hyperbolic cosine of Z1.

"HTAN": Compute the hyperbolic tangent of Z1.

"HSEC": Compute the hyperbolic secant of Z1.

"HCSC": Compute the hyperbolic cosecant of Z1.

"HCOT": Compute the hyperbolic cotangent of Z1.

"#SIN": Inverse hyperbolic sine of Z1.

"#COS": Inverse hyperbolic cosine of Z1.

"#TAN": Inverse hyperbolic tangent of Z1.

"#SEC": Inverse hyperbolic secant of Z1.

"#CSC": Inverse hyperbolic cosecant of Z1. Result in degrees.

"#COT": Inverse hyperbolic contangent of Z1. Result is given in degrees.

"SZ": Switch Z1 and Z1 registers. (Z1➔Z2➔Z1).

"RZ": Rotate the Z registers. Z1 is put in Z2, Z2 in Z3 etc. (Z1➔Z2➔Z3➔Z4➔Z1).

"QUAD": Compute solution to quadratic equation ax 2 + bx + c. a =Z3 b= Z2 c=Z1. Results in Z1 and Z2.

"SB": Escape to SuperBasic for a time. You return to the program with CONTINUE. Z and M registers are unchanged. In SB you may calculate, e.g. "PRINT 34+45".

"STO": Store Z1 in Memory (1-9). You are asked which memory. You may add a note against the number to help you remember what is saved.

"SUM": Add Z1 to Memory (1-9). You are asked on which Memory you want to work.

"RCL": Recall the value from the Memory location you chose and put it in Z1 for use in subsequent calculation.

"CMs": Clear chosen Memory or all Memories. You are asked which.

"HELP": Gives HELP on the Command/Function you type after "HELP". If no Command/Function is entered, "HELP" will give help on calculations in general.

REVERSE POLISH NOTATION CALCULATOR

*	+	/	–		INV	NEG	RO	CLR	SIN	ASIN	COS	ACOS	TAN
ATAN	SEC	ASEC	COT	ACOT	CSEC	ACSC	PI	LN	E	LOG	%	N!	STAT
HYP	SIDE	DEG	RAD	POLR	RECT	MET	HSIN	HCOS	HTAN	HSEC	HSCS	HCOT	#SIN
#COS	#TAN	#SEC	#CSC	#COT	SZ	RZ	QUAD	SB	STO	SUM	RCL	CMs	HELP

```
100 REMark                Scientific
110 REMark    Reverse Polish Notation Calculator
120 REMark          Peter Lauridsen, 1988
130 :
140 initialize : window_7 : window_3 : window_4
150 window_5 : window_6 : menu
160 :
170 REPeat loop
180   CLS#6
190   AT#4,1,17 : PRINT#4,'                    '
200   AT#4,1,2 : INPUT#4, 'Command/value: ';c$
210   AT#4,2,5
220   PRINT#4,'
230   IF c$='' : GO TO 180
240   IF c$=='Quit' : quit
250   number_or_command
260   END REPeat loop
270 :
280 DEFine PROCedure number_or_command
290   IF NOT c$(1) INSTR'.0123456789'
300     command : RETurn
310     END IF
320   x=c$ : number_count : shift_registers
330 END DEFine
340 :
350 DEFine PROCedure command
360   IF NOT c$ INSTR'+*-/'
370       find_command_procedure
380     ELSE
390       command_count
400     END IF
410   END DEFine
420 :
430 DEFine PROCedure number_count
440   IF n(1)=2
450       n(2)=2
460     ELSE
470       n(1)=2
480     END IF
490   END DEFine
500 :
510 DEFine PROCedure command_count
520   IF n(1)=1
530       n(2)=1
540     ELSE
550       n(1)=1
560     END IF
570   find_command_procedure
580   END DEFine
590 :
600 DEFine PROCedure find_command_procedure
610   IF c$= '*'    : multiply : RETurn
620   IF c$= '+'    : addition : RETurn
630   IF c$= '/'    : divide   : RETurn
640   IF c$= '-'    : subtract : RETurn
650   IF c$= '^'    : powers   : RETurn
660   IF c$=='INV'  : inverse : RETurn
670   IF c$=='NEG'  : negative : RETurn
680   IF c$=='RO'   : round_off : RETurn
690   IF c$=='CLR'  : clr_registers : RETurn
700   IF c$=='SIN'  : sine : RETurn
710   IF c$=='ASIN' : arc_sine : RETurn
720   IF c$=='COS'  : cosine : RETurn
730   IF c$=='ACOS' : arc_cosine : RETurn
740   IF c$=='TAN'  : tangent : RETurn
750   IF c$=='ATAN' : arc_tangent : RETurn
760   IF c$=='SEC'  : secant : RETurn
770   IF c$=='ASEC' : arc_secant : RETurn
780   IF c$=='COT'  : cotangent : RETurn
790   IF c$=='ACOT' : arc_cotangent : RETurn
800   IF c$=='CSEC' : cosecant : RETurn
810   IF c$=='ACSC' : arc_cosecant : RETurn
820   IF c$=='PI'   : phi : RETurn
830   IF c$=='LN'   : natural_logarithm: RETurn
840   IF c$=='E'    : exponential_of_x : RETurn
850   IF c$=='LOG'  : logarithm : RETurn
860   IF c$= '%'    : percent : RETurn
870   IF c$=='N!'   : factorial : RETurn
880   IF c$=='STAT' : statistics : RETurn
890   IF c$=='HYP'  : hypotenuse : RETurn
900   IF c$=='SIDE' : side_of_triangle : RETurn
910   IF c$=='DEG'  : conv_to_degrees : RETurn
920   IF c$=='RAD'  : conv_to_radians : RETurn
930   IF c$=='POLR' : rect_to_polar_conv : RETurn
940   IF c$=='RECT' : polar_to_rect_conv : RETurn
950   IF c$=='MET'  : conv_to_metric : RETurn
960   IF c$=='HSIN' : hyperbolic_sine : RETurn
970   IF c$=='HCOS' : hyperbolic_cosine : RETurn
980   IF c$=='HTAN' : hyperbolic_tangent : RETurn
990   IF c$=='HSEC' : hyperbolic_secant : RETurn
1000  IF c$=='HCSC' : hyperbolic_cosecant: RETurn
1010  IF c$=='HCOT' : hyperbolic_cotangent:RETurn
1020  IF c$=='#SIN' : inv_hyperbolic_sine : RETurn
1030  IF c$=='#COS' : inv_hyperbolic_cosine:RETurn
1040  IF c$=='#TAN' :inv_hyperbolic_tangent:RETurn
1050  IF c$=='#SEC' : inv_hyperbolic_secant:RETurn
1060  IF c$=='#CSC' :inv_hyperbolic_cosecant:RETurn
1070 IF c$=='#COT' :inv_hyperbolic_cotangent:RETurn
1080  IF c$=='SZ'   : switch_to_z1 : RETurn
1090  IF c$=='RZ'   : rotate_z : RETurn
1100  IF c$=='QUAD' : quadratic_equation : RETurn
1110  IF c$=='SB'   : monitor_mode : RETurn
1120  IF c$=='STO'  : store_in_memory : RETurn
1130  IF c$=='SUM'  : add_to_memory : RETurn
1140  IF c$=='RCL'  : recall_memory : RETurn
1150  IF c$=='CMs'  : clear_memory : RETurn
1160  IF c$=='HELP' : help : RETurn
1170  error_message
1180  END DEFine
1190 :
1200 DEFine PROCedure multiply
1210   IF n(1)=n(2) AND n(1)=1
1220       x=z1*z3 : n(2)=2
1230     ELSE
1240       x=z1*z2
1250     END IF
1260   shift_registers
1270   END DEFine
1280 :
1290 DEFine PROCedure addition
1300   IF n(1)=1 AND n(1)=n(2)
1310       x=z1+z3 : n(2)=2
1320     ELSE
1330       x=z1+z2
1340     END IF
1350   shift_registers
1360   END DEFine
1370 :
1380 DEFine PROCedure divide
1390   IF z1=0 : error_message : RETurn
1400   IF n(1)=n(2) AND n(1)=1
1410       x=z3/z1 : n(2)=2
1420     ELSE
1430       x=z2/z1
1440     END IF
1450   shift_registers
1460   END DEFine
1470 :
1480 DEFine PROCedure subtract
1490   IF n(1)=n(2) AND n(1)=1
1500       x=z3-z1 : n(2)=2
1510     ELSE
1520       x=z2-z1
1530     END IF
1540   shift_registers
1550   END DEFine
1560 :
1570 DEFine PROCedure powers
1580   x=z2^z1 : shift_registers
1590   END DEFine
1600 :
1610 DEFine PROCedure inverse
1620   IF z1=0 : error_message : RETurn
1630   x=1/z1 : put_in_z1
1640   END DEFine
1650 :
1660 DEFine PROCedure negative
1670   x=-z1 : put in z1
```

```
1680   END DEFine
1690 :
1700 DEFine PROCedure round_off
1710 AT #6,1,20
1720 PRINT#6,'How many decimals (0-6):
1730   a$=INKEY$(-1)
1740   IF NOT a$ INSTR'0123456'
1750     error_message : GO TO 1710
1760     END IF
1770   places=a$
1780   x=(INT(z1*(10^places)+.5))/10^places
1790   CLS#6 : put_in_z1
1800   END DEFine
1810 :
1820 DEFine PROCedure clr_registers
1830   z1=0 : z2=0 : z3=0 : z4=0
1840   n(1)=1 : n(2)=1 : print_registers
1850   END DEFine
1860 :
1870 DEFine PROCedure sine
1880   IF z1>60000 OR z1<-60000
1890     error_message : RETurn
1900     END IF
1910   x=SIN(z1*1.74533E-2)
1920   put_in_z1
1930   END DEFine
1940 :
1950 DEFine PROCedure arc_sine
1960   IF z1>1 OR z1<-1 : error_message : RETurn
1970   x=ASIN(z1)*57.29578 : put_in_z1
1980   END DEFine
1990 :
2000 DEFine PROCedure cosine
2010   IF z1>347745 OR z1<-347745
2020     error_message : RETurn
2030     END IF
2040   x=COS(z1*1.74533E-2) : put_in_z1
2050   END DEFine
2060 :
2070 DEFine PROCedure arc_cosine
2080   IF z1>1 OR z1<-1 : error_message : RETurn
2090   x=ACOS(z1)*57.29578 : put_in_z1
2100   END DEFine
2110 :
2120 DEFine PROCedure tangent
2130   IF z1>30000 OR z1<-30000
2140     error_message : RETurn
2150     END IF
2160   x=TAN(z1*1.74533E-2) : put_in_z1
2170   END DEFine
2180 :
2190 DEFine PROCedure arc_tangent
2200   x=ATAN(z1)*57.29578 : put_in_z1
2210   END DEFine
2220 :
2230 DEFine PROCedure secant
2240   IF z1>347745 OR z1<-3.437745E6
2250     error_message : RETurn
2260     END IF
2270   t=COS(z1*1.745333E-2)
2280   IF t=0 : error_message : RETurn
2290   x=1/COS(z1*1.74533E-2) : put_in_z1
2300   END DEFine
2310 :
2320 DEFine PROCedure arc_secant
2330   part1=ATAN(SQRT(z1*z1-1))
2340   p=(z1-1)*1.5708 : part2 = sgn(p)
2350   x=(part1+part2)*57.29578 : put_in_z1
2360   END DEFine
2370 :
2380 DEFine PROCedure cotangent
2390   IF z1=1.718872E6 OR z1=-1.718872E6
2400     error_message : RETurn
2410     END IF
2420   x=COT(z1*1.74533E-2) : put_in_z1
2430   END DEFine
2440 :
2450 DEFine PROCedure arc_cotangent
2460   x=ACOT(z1)*57.29578 : put_in_z1
2470   END DEFine
2480 :
2490 DEFine PROCedure cosecant
2500   t=SIN(z1*1.74533E-2)
2510   IF t=0 : x=1E616 : put_in_z1 : RETurn
2520   x=1/SIN(z1*1.74533E-2) : put_in_z1
2530   END DEFine
2540 :
2550 DEFine PROCedure arc_cosecant
2560   IF z1=-1 AND z1=1 : error_message : RETurn
2570   IF z1=1 : x=90 : put_in_z1 : RETurn
2580   part1=ATAN(1/SQRT(z1*z1-1))
2590   part2=sgn(z1)-1 : part2=part2*1.5708
2600   x=(part1+part2)*57.29578 : put_in_z1
2610   END DEFine
2620 :
2630 DEFine PROCedure phi
2640   x=3.141593 : number_count : shift_registers
2650   END DEFine
2660 :
2670 DEFine PROCedure natural_logarithm
2680   IF x<=0 : error_message : RETurn
2690   x=LN(z1) : put_in_z1
2700   END DEFine
2710 :
2720 DEFine PROCedure exponential_of_x
2730   IF z1>500 OR z1<-500
2740     error_message : RETurn
2750     END IF
2760   x=EXP(z1) : put_in_z1
2770   END DEFine
2780 :
2790 DEFine PROCedure logarithm
2800   IF z1<=0 : error_message : RETurn
2810   x=LOG10(z1) : put_in_z1
2820   END DEFine
2830 :
2840 DEFine PROCedure percent
2850   x=z1*1E-2 : put_in_z1
2860   END DEFine
2870 :
2880 DEFine PROCedure factorial
2890   IF z1=0 : z1=1 : GO TO 2940
2900   x=z1
2910   FOR y=1 TO z1-1
2920     x=x*(z1-y)
2930     END FOR y
2940   put_in_z1
2950   END DEFine
2960 :
2970 DEFine PROCedure statistics
2980   CLS#3 : CLS#5
2990   PRINT#3, '                          In Statisti
cal Mode'\\\
3000   PRINT#3, '       Enter values separately an
d enter "Z" when done'
3010   sum=0 : ms=0 : d=1
3020   INPUT#5, a$
3030   FOR p=1 TO LEN(a$)
3040     IF NOT a$(p) INSTR'zZ1234567890'
3050       GO TO 3020
3060       END IF
3070     END FOR p
3080   IF a$='' THEN  GO TO 3020
3090   IF a$(1)=='z' THEN stat_results : RETurn
3100   k(d)=a$ : sum=sum+k(d)
3110   ms=ms+((k(d))^2) : d=d+1 : GO TO 3020
3120   END DEFine
3130 :
3140 DEFine PROCedure stat_results
3150   CLS#3 : CLS#5 : PRINT#3, 'Scores:
3160   IF d=1 OR d=2: menu: error_message: RETurn
3170   FOR h=1 TO d-1
3180     PRINT#3, k(h),
3190     END FOR h
3200   AT#5,1,0
3210   PRINT #5, 'Sum of scores:        ', sum
3220   PRINT #5, 'Number of scores:     ', d-1
3230   PRINT #5, 'Mean value:           ', sum/(d-1)
3240   PRINT #5, 'Sum mean squared:     ', ms
3250   PRINT #5, 'Variance:             ', (ms/(d-1)
)-((sum/(d-1))^2)
3260   PRINT #5, 'Standard deviation: ', SQRT((ms-
((sum^2)/(d-1)))/(d-2))
3270   PRINT #5\\
3280   PRINT #5, 'Press a KEY to continue'
3290   a$=INKEY$(-1) : CLS#3 : CLS#5
3300   menu
3310   print_memories
3320   END DEFine
3330 :
3340 DEFine PROCedure hypotenuse
3350   IF z1<0 OR z2<0 : error_message : RETurn
3360   x=SQRT(z1^2+z2^2) : shift_registers
3370   END DEFine
3380 :
3390 DEFine PROCedure side_of_triangle
3400   IF z1<0 OR z2<0 : error_message : RETurn
3410   x=SQRT(ABS(z1^2-z2^2)) : shift_registers
3420   END DEFine
3430 :
```

```
3440 DEFine PROCedure conv_to_degrees
3450   x=z1*57.29578 : put_in_z1
3460   END DEFine
3470 :
3480 DEFine PROCedure conv_to_radians
3490   x=1.745329E-2*z1 : put_in_z1
3500   END DEFine
3510 :
3520 DEFine PROCedure rect_to_polar_conv
3530   b=(z1*z1+z2*z2)
3540   IF b<=0 : error_message : RETurn
3550   z2=SQRT(z1*z1+z2*z2) : p=z1/z2
3560   z1=(ATAN(p/SQRT(-p*p+1)))*57.29578
3570   print_registers
3580   END DEFine
3590 :
3600 DEFine PROCedure polar_to_rect_conv
3610   IF z1>60000 OR z1<-60000 OR z2>60000 OR z2<
-60000
3620     error_message : RETurn
3630     END IF
3640   b=z2*(SIN(z1*1.745329E-2))
3650   z2=z2*(COS(z1*1.745329E-2))
3660   z1=b : print_registers
3670   END DEFine
3680 :
3690 DEFine PROCedure conv_to_metric
3700   CLS#5
3710   PRINT#5, '  C O N V E R T:'\
3720   PRINT#5, '  1) feet to meters'
3730   PRINT#5, '  2) inches to centimeters'
3740   PRINT#5, '  3) miles to kilometers'
3750   PRINT#5, '  4) gallons to liters'
3760   PRINT#5, '  5) fahrenheit to centigrade'
3770   PRINT#5, '  6) pounds to kilograms'\\
3780   PRINT#5, '  Use the negative to convert'
3790   PRINT#5, '  vice-versa'
3800   AT#6,1,20 : INPUT#6, 'Select: ',h$
3810   IF NOT h$(LEN(h$)) INSTR '123456'
3820     AT#6,1,30 : PRINT#6,'          '
3830     GO TO 3700
3840     END IF
3850   h=h$ : CLS#5 : CLS#5
3860   print_memories
3870   :
3880   IF h=1 : feet_to_meters : RETurn
3890   IF h=2 : inches_to_cm : RETurn
3900   IF h=3 : miles_to_km : RETurn
3910   IF h=4 : gallons_to_liters : RETurn
3920   IF h=5 : fahrenheit_to_centigrades : RETurn
3930   IF h=6 : pounds_to_kg : RETurn
3940   IF h=-1 : meters_to_feet : RETurn
3950   IF h=-2 : cm_to_inches : RETurn
3960   IF h=-3 : km_to_miles : RETurn
3970   IF h=-4 : liters_to_gallons : RETurn
3980   IF h=-5 : centigrades_to_fahrenheit: RETurn
3990   IF h=-6 : kg_to_pounds : RETurn
4000   GO TO 3700
4010   END DEFine
4020 :
4030 DEFine PROCedure feet_to_meters
4040   x=.3048 * z1 : result
4050   END DEFine
4060 :
4070 DEFine PROCedure meters_to_feet
4080   x=z1 * 3.28084 : result
4090   END DEFine
4100 :
4110 DEFine PROCedure inches_to_cm
4120   x=z1 * 2.54 : result
4130   END DEFine
4140 :
4150 DEFine PROCedure cm_to_inches
4160   x=z1 * .3937008 : result
4170   END DEFine
4180 :
4190 DEFine PROCedure miles_to_km
4200   x=z1 * 1.609344 : result
4210   END DEFine
4220 :
4230 DEFine PROCedure km_to_miles
4240   x=z1 * .6213711 : result
4250   END DEFine
4260 :
4270 DEFine PROCedure gallons_to_liters
4280   x=3.785412 * z1 : result
4290   END DEFine
4300 :
4310 DEFine PROCedure liters_to_gallons
4320   x=.2641721 * z1 : result
4330   END DEFine
4340 :
4350 DEFine PROCedure fahrenheit_to_centigrades
4360   x=.5555555 * (z1-32) : result
4370   END DEFine
4380 :
4390 DEFine PROCedure centigrades_to_fahrenheit
4400   x=(1.8 * z1) + 32 : result
4410   END DEFine
4420 :
4430 DEFine PROCedure pounds_to_kg
4440   x=.4535924 * z1 : result
4450   END DEFine
4460 :
4470 DEFine PROCedure kg_to_pounds
4480   x=2.204622 * z1 : result
4490   END DEFine
4500 :
4510 DEFine PROCedure hyperbolic_sine
4520   IF z1>500 OR z1<0 : error_message : RETurn
4530   x=(EXP(z1)-EXP(-z1))/2 : put_in_z1
4540   END DEFine
4550 :
4560 DEFine PROCedure hyperbolic_cosine
4570   IF z1>500 OR z1<0: error_message: RETurn
4580   x=(EXP(z1)+EXP(-z1))/2 : put_in_z1
4590   END DEFine
4600 :
4610 DEFine PROCedure hyperbolic_tangent
4620   IF z1>500 OR z1<0: error_message: RETurn
4630   x=(EXP(z1)-EXP(-z1))/(EXP(z1)+EXP(-z1))
4640   put_in_z1
4650   END DEFine
4660 :
4670 DEFine PROCedure hyperbolic_secant
4680   x=2/(EXP(z1)+EXP(-z1)) : put_in_z1
4690   END DEFine
4700 :
4710 DEFine PROCedure hyperbolic_cosecant
4720   x=2/(EXP(z1)-EXP(-z1)) : put_in_z1
4730   END DEFine
4740 :
4750 DEFine PROCedure hyperbolic_cotangent
4760   x=EXP(-z1)/(EXP(z1)-EXP(-z1))*2+1
4770   put_in_z1
4780   END DEFine
4790 :
4800 DEFine PROCedure inv_hyperbolic_sine
4810   x=LN(z1+SQRT(z1*z1+1)) : put_in_z1
4820   END DEFine
4830 :
4840 DEFine PROCedure inv_hyperbolic_cosine
4850   IF z1<=1 : error_message : RETurn
4860   x=LN(z1+SQRT(z1*z1-1)) : put_in_z1
4870   END DEFine
4880 :
4890 DEFine PROCedure inv_hyperbolic_tangent
4900   IF z1>=1 OR z1<0 : error_message : RETurn
4910   x=LN((1+z1)/(1-z1))/2 : put_in_z1
4920   END DEFine
4930 :
4940 DEFine PROCedure inv_hyperbolic_secant
4950   IF z1<=0 : error_message : RETurn
4960   x=LN((SQRT(-z1*z1+1)+1)/z1) : put_in_z1
4970   END DEFine
4980 :
4990 DEFine PROCedure inv_hyperbolic_cosecant
5000   IF z1=0 : error_message : RETurn
5010   x=LN((sgn(z1)*SQRT(z1*z1+1)+1)/z1)
5020   put_in_z1
5030   END DEFine
5040 :
5050 DEFine PROCedure inv_hyperbolic_cotangent
5060   t=z1-1 : IF t<=0 : error_message : RETurn
5070   x=LN((z1+1)/(z1-1))/2 : put_in_z1
5080   END DEFine
5090 :
5100 DEFine PROCedure switch_to_z1
5110   b=z2: z2=z1: z1=b: n(2)=2: print_registers
5120   END DEFine
5130 :
5140 DEFine PROCedure rotate_z
5150   b=z4: z4=z3: z3=z2: z2=z1: z1=b: n(1)=2
5160   print_registers
5170   END DEFine
5180 :
5190 DEFine PROCedure quadratic_equation
5200   IF z3=0 : error_message : RETurn
5210   t=z2*z2-4*z3*z1
5220   IF t<0 : error_message : RETurn
```

```
5230    b=(-z2+(SQRT(z2*z2-4*z3*z1)))/(2*z3)
5240    z1=(-z2-(SQRT(z2*z2-4*z3*z1)))/(2*z3)
5250    z2=b : print_registers
5260    END DEFine
5270 :
5280 DEFine PROCedure monitor_mode
5290    CLS#1 : PRINT#1, 'Entering monitor mode;';
5300    PRINT#1,' type CONTINUE and press ENTER ';
5310    PRINT#1,'to continue...'
5320    CLS#0 : STOP : CLS#0
5330    window_7 : window_3 : window_4
5340    window_5 : window_6 : menu
5350    print_registers : GO TO 200
5360    END DEFine
5370 :
5380 DEFine PROCedure store_in_memory
5390    AT#6,0,15
5400    PRINT#6,'Which memory location? (1-9): ';
5410    INPUT#6,mem
5420    AT#6,1,15 : PRINT#6,'Remark on memory No.';
5430    PRINT#6, mem;'          : ';
5440    INPUT#6,note$(mem)
5450    memory(mem)=z1 : CLS#6 : print_memories
5460    END DEFine
5470 :
5480 DEFine PROCedure add_to_memory
5490    AT#6,1,20: PRINT#6,'Which memory? (1-9): ';
5500    INPUT#6,mem
5510    memory(mem)=memory(mem)+z1 : CLS#6
5520    print_memories
5530    END DEFine
5540 :
5550 DEFine PROCedure recall_memory
5560    AT#6,1,20: PRINT#6,'Which memory? (1-9): ';
5570    INPUT#6,mem : x=memory(mem): number_count
5580    CLS#6 : shift_registers
5590    END DEFine
5600 :
5610 DEFine PROCedure clear_memory
5620    AT#6,1,10
5630    PRINT#6,'Which memory (1-9;0 clears all):';
5640    mem=INKEY$(-1)
5650    IF mem=0
5660      FOR x=1 TO 9 : memory(x)=0 : note$(x)=''
5670      END FOR x
5680      CLS#5 : CLS#6 : RETurn
5690    ELSE
5700      memory(mem)=0 : note$(mem)=''
5710      END IF
5720    CLS#6 : print_memories
5730    END DEFine
5740 :
5750 DEFine PROCedure print_memories
5760    FOR x=1 TO 9
5770      AT#5,x,5
5780      PRINT#5,'
5790      END FOR x
5800    FOR x=1 TO 9
5810      AT#5,x,2 : PRINT#5,'M';x;': ';memory(x)
5820      AT#5,x,18 : PRINT#5, note$(x)
5830      END FOR x
5840    END DEFine
5850 :
5860 DEFine PROCedure help
5870    CLS#6
5880    AT#6,1,20:PRINT#6,'HELP on what function?';
5890    INPUT#6,c$
5900    IF c$= ''
5910      write text$(0) : write text$(1)
5920      write text$(2) : write text$(3) : RETurn
5930      END IF
5940    IF c$= '*'     : write text$(7)  : RETurn
5950    IF c$= '+'     : write text$(7)  : RETurn
5960    IF c$= '/'     : write text$(7)  : RETurn
5970    IF c$= '-'     : write text$(7)  : RETurn
5980    IF c$= '^'     : write text$(8)  : RETurn
5990    IF c$=='INV'   : write text$(9)  : RETurn
6000    IF c$=='NEG'   : write text$(10) : RETurn
6010    IF c$=='RO'    : write text$(11) : RETurn
6020    IF c$=='CLR'   : write text$(12) : RETurn
6030    IF c$=='SIN'   : write text$(13) : RETurn
6040    IF c$=='ASIN'  : write text$(14) : RETurn
6050    IF c$=='COS'   : write text$(15) : RETurn
6060    IF c$=='ACOS'  : write text$(16) : RETurn
6070    IF c$=='TAN'   : write text$(17) : RETurn
6080    IF c$=='ATAN'  : write text$(18) : RETurn
6090    IF c$=='SEC'   : write text$(19) : RETurn
6100    IF c$=='ASEC'  : write text$(20) : RETurn
6110    IF c$=='COT'   : write text$(21) : RETurn
6120    IF c$=='ACOT'  : write text$(22) : RETurn
6130    IF c$=='CSEC'  : write text$(23) : RETurn
6140    IF c$=='ACSC'  : write text$(24) : RETurn
6150    IF c$=='PI'    : write text$(25) : RETurn
6160    IF c$=='LN'    : write text$(26) : RETurn
6170    IF c$=='E'     : write text$(27) : RETurn
6180    IF c$=='LOG'   : write text$(28) : RETurn
6190    IF c$= '%'     : write text$(29) : RETurn
6200    IF c$=='N!'    : write text$(30) : RETurn
6210    IF c$=='STAT'  : write text$(31) : RETurn
6220    IF c$=='HYP'   : write text$(32) : RETurn
6230    IF c$=='SIDE'  : write text$(33) : RETurn
6240    IF c$=='DEG'   : write text$(34) : RETurn
6250    IF c$=='RAD'   : write text$(35) : RETurn
6260    IF c$=='POLR'  : write text$(36) : RETurn
6270    IF c$=='RECT'  : write text$(37) : RETurn
6280    IF c$=='MET'   : write text$(38) : RETurn
6290    IF c$=='HSIN'  : write text$(39) : RETurn
6300    IF c$=='HCOS'  : write text$(40) : RETurn
6310    IF c$=='HTAN'  : write text$(41) : RETurn
6320    IF c$=='HSEC'  : write text$(42) : RETurn
6330    IF c$=='HCSC'  : write text$(43) : RETurn
6340    IF c$=='HCOT'  : write text$(44) : RETurn
6350    IF c$=='#SIN'  : write text$(45) : RETurn
6360    IF c$=='#COS'  : write text$(46) : RETurn
6370    IF c$=='#TAN'  : write text$(47) : RETurn
6380    IF c$=='#SEC'  : write text$(48) : RETurn
6390    IF c$=='#CSC'  : write text$(49) : RETurn
6400    IF c$=='#COT'  : write text$(50) : RETurn
6410    IF c$=='SZ'    : write text$(51) : RETurn
6420    IF c$=='RZ'    : write text$(52) : RETurn
6430    IF c$=='QUAD'  : write text$(53) : RETurn
6440    IF c$=='SB'    : write text$(54) : RETurn
6450    IF c$=='STO'   : write text$(55) : RETurn
6460    IF c$=='SUM'   : write text$(56) : RETurn
6470    IF c$=='RCL'   : write text$(57) : RETurn
6480    IF c$=='CMs'   : write text$(58) : RETurn
6490    IF c$=='HELP'  : write text$(59) : RETurn
6500    CLS#6 : error_message
6510    END DEFine
6520 :
6530 DEFine PROCedure put_in_z1
6540    z1=x : n(1)=2 : n(2)=2 : print_registers
6550    END DEFine
6560 :
6570 DEFine PROCedure shift_registers
6580    z4=z3 : z3=z2 : z2=z1 : z1=x
6590    print_registers
6600    END DEFine
6610 :
6620 DEFine PROCedure print_registers
6630    AT#4,2,0
6640    PRINT#4,'
6650    AT#4,3,13: PRINT#4,'Z1: ';z1;'
6660    AT#4,4,13: PRINT#4,'Z2: ';z2;'
6670    AT#4,5,13: PRINT#4,'Z3: ';z3;'
6680    AT#4,6,13: PRINT#4,'Z4: ';z4;'
6690    END DEFine
6700 :
6710 DEFine PROCedure result
6720    CLS#4 : put_in_z1
6730    END DEFine
6740 :
6750 DEFine PROCedure menu
6760    CLS#3
6770    PRINT#3,'                          REVERSE POLISH
 NOTATION CALCULATOR'
6780    PRINT#3,'  *     +     /     -          INV  NE
G  RO   CLR  SIN  ASIN COS  ACOS  TAN '
6790    PRINT #3,'  ATAN SEC  ASEC COT  ACOT CSEC A
CSC PI   LN   E    LOG  %    N!   STAT'
6800    PRINT #3,'  HYP  SIDE DEG  RAD  POLR RECT M
ET  HSIN HCOS HTAN HSEC HCSC HCOT #SIN'
6810    PRINT#3, '  #COS #TAN #SEC #CSC #COT SZ   R
Z   QUAD SB   STO  SUM  RCL  CMs  HELP'
6820    PRINT#3,'                                  Use QU
IT to end'
6830    END DEFine
6840 :
6850 DEFine PROCedure initialize
6860    MODE 4: CLS#0
6870 DIM a$(14),c$(14),k(25),memory(9),note$(9,14)
6880    DIM h$(10),text$(60,200),n(2),tekst$(200)
6890    z1=0 : z2=0 : z3=0 : z4=0 : c$='': n(1)=1
6900    FOR x=1 TO 6
6910      memory(x)=0 : note$(x,14)=''
6930      END FOR x
6940    help_text
6950    END DEFine
6960 :
6970 DEFine PROCedure quit
```

```
6980    CLOSE#3 : CLOSE#4 : CLOSE#5 : CLOSE#6
6990    OPEN#7,scr_512x256a0x0 : CLS#7 : CLOSE#7
7000    CLS#0 : CLS#1 : CLS#2 : NEW
7010    END DEFine
7020 :
7030 DEFine FuNction sgn(p)
7040    RETurn p>0
7050    END DEFine
7060 :
7070 DEFine PROCedure window_3
7080    OPEN #3,con_448x64a32x16 : PAPER #3,2
7090    INK #3,7 : BORDER #3,2,0 : CLS#3
7100    END DEFine
7110 :
7120 DEFine PROCedure window_4
7130    OPEN #4,con_218x112a32x86 : PAPER #4,7
7140    INK #4,0 : BORDER#4,2,0 : CLS#4
7150    END DEFine
7160 :
7170 DEFine PROCedure window_5
7180    OPEN #5,con_218x112a262x86 : PAPER #5,7
7190    INK #5,0 : BORDER #5,2,0 : CLS#5
7200    END DEFine
7210 :
7220 DEFine PROCedure window_6
7230    OPEN #6,con_448x36a32x206 : PAPER #6,2
7240    INK #6,7 : BORDER #6,2,0 : CLS#6
7250    END DEFine
7260 :
7270 DEFine PROCedure window_7
7280    OPEN#7,scr_512x256a0x0 : PAPER #7,4: CLS #7
7290    BLOCK#7,448,64,38,20,4,0
7300    BLOCK#7,218,112,38,90,4,0
7310    BLOCK#7,218,112,268,90,4,0
7320    BLOCK#7,448,36,38,210,4,0 : CLOSE #7
7330    END DEFine
7340 :
7350 DEFine PROCedure error_message
7360    BEEP 2000,50
7370    AT#4,2,17 : PRINT#4,'ILLEGAL FUNCTION'
7380    END DEFine
7390 :
7400 DEFine PROCedure help_text
7410    text$(0)=' The Program Emulates a Reverse P
olish Notation Scientific Calculator.    Z1-Z4 are
 registers. M1-M9 are memories.'
7420    text$(1)=' No parenthesis and no "=" keys a
re used. Only two  numbers  are  worked  with at a
 time. If two numbers are entered and "+" is press
ed Z2 and Z1  will be added.'
7430    text$(2)=' If a new number is entered and "
+" is pressed, Z2 and Z1 will be added.  If "+" is
 pressed again, the result in Z3 and Z1 will be ad
ded.'
7440    text$(3)=' After a conversion the calculato
r will work on Z2 and Z1.'
7450    text$(7)=' * + / - are used to multiply, ad
d, divide and subtract Z1 and Z2(or Z3). See help
on "". ("HELP", <ENTER>,<ENTER>).'
7460    text$(8)=' "^": Raises the number in Z2 (or
 Z3) to the power of the value in Z1.'
7470    text$(9)=' "INV": Calculates the inverse va
lue (1/x) of the number in Z1.           1/Z1 Z1'
7480    text$(10)=' "NEG": Negate. Multiply the num
ber in Z1 with -1.                       Z1  Z1*(
-1).'
7490    text$(11)=' "RO": Round off the value in Z1
 to a chosen number of digits.            Eg. 2.34
5  "RO" 2 <ENTER>  2.35'
7500    text$(12)=' "CLR" : Clears all Z-registers
of their content. Warning:  Values  not   stored i
n Memories are lost.'
7510    text$(13)=' "SIN" : Compute the sine of Z1
which is taken to be in degrees.'
7520    text$(14)=' "ASIN" : Compute the arcsine of
 Z1 in degrees.'
7530    text$(15)=' "COS" : Compute the cosine of Z
1 which is taken s degrees.'
7540    text$(16)=' "ACOS" : Compute the arccosine
of Z1 in degrees.'
7550    text$(17)=' "TAN"  : Compute the tangent of
 Z1 which is taken to be in degrees.'
7560    text$(18)=' "ATAN" : Compute the arctangent
 of Z1. Result in degrees.'
7570    text$(19)=' "SEC" : Compute the secant of Z
1 which is taken to be in degrees.'
7580    text$(20)=' "ASEC" : Compute the arcsecant
of Z1. Result in degrees.'
7590    text$(21)=' "COT" : Compute the cotangent o
f Z1 which is taken to be in degrees.'
7600    text$(22)=' "ACOT" : Compute the arccotange
nt of Z1. Result in degrees.'
7610    text$(23)=' "CSC" : Compute the cosecant of
 Z1 which is taken to be in degrees.'
7620    text$(24)=' "ACSC" : Compute the arccosecan
t of Z1. Result in degrees.'
7630    text$(25)=' "PI" : Put the value or  in Z1
. =3.141593'
7640    text$(26)=' "LN" : Compute the Natural Loga
rithm of Z1. Base e=2.7182818'
7650    text$(27)=' "E" : Compute the exponential o
f Z1. (e^X).'
7660    text$(28)=' "LOG" : Compute the logarithm o
f Z1.'
7670    text$(29)=' "%" : Multiply Z1 with 0.01.'
7680    text$(30)=' "N!" : Compute the faculty of Z
1. Eg. 4! = 4*3*2*1 = 24'
7690    text$(31)=' "STAT" : Subprogram. Compute th
e statistical data of a set to. observa- tions. Y
ou will have to write down the results, as they ar
e  lost  when  you return to the calculator.'
7700    text$(32)=' "HYP" : Compute the value for t
he hypoteneuse of a triangle, given two sides.'
7710    text$(33)=' "SIDE": Compute the value for a
 side of a triangle given the hypotenuse   and oth
er side. Z1(side1),Z2(hyp)  Z1(side2),Z2(side1),Z
3(hyp).'
7720    text$(34)=' "DEG" : Change from radians to
degrees. Z1(rad)  Z1(deg).'
7730    text$(35)=' "RAD" : Change from degrees to
radians.'
7740    text$(36)=' "POLR" : Compute polar coordina
tes given rectangular X=Z2 Y=Z1  R=Z2 <=Z1.'
7750    text$(37)=' "RECT" : Compute rectangular co
ordinates given polar R=Z2 <=Z1  X=Z2 Y=Z1.'
7760    text$(38)=' "MET" : Compute metric conversi
ons.'
7770    text$(39)=' "HSIN" : Compute the hyperbolic
 sine of Z1.'
7780    text$(40)=' "HCOS" : Compute the hyperbolic
 cosine of Z1.'
7790    text$(41)=' "HTAN" : Compute the hyperbolic
 tangent of Z1.'
7800    text$(42)=' "HSEC" : Compute the hyperbolic
 secant of Z1.'
7810    text$(43)=' "HCSC" : Compute the hyperbolic
 cosecant of Z1.'
7820    text$(44)=' "HCOT" : Compute the hyperbolic
 cotangent of Z1.'
7830    text$(45)=' "#SIN" : Inverse hyperbolic sin
e of Z1.'
7840    text$(46)=' "#COS" : Inverse hyperbolic cos
ine of Z1.'
7850    text$(47)=' "#TAN" : Inverse hyperbolic tan
gent of Z1.'
7860    text$(48)=' "#SEC" : Inverse hyperbolic sec
ant of Z1.'
7870    text$(49)=' "#CSC" : Inverse hyperbolic cos
ecant of Z1. Result in degrees'
7880    text$(50)=' "#COT" : Inverse hyperbolic cot
angent of Z1. Result is given in degrees.'
7890    text$(51)=' "SZ" : Switch Z1 and Z1 registe
rs.( Z1Z2 Z1 ).'
7900    text$(52)=' "RZ" : Rotate the Z registers.
Z1 is put in Z2, Z2 in Z3 etc.            (Z1Z2
Z3Z4  Z1).'
7910    text$(53)=' "QUAD" : Compute solution to qu
adratic equation ax^2 + bx + c.                  a=Z
3 b=Z2  c=Z1. Results in Z1 and Z2.'
7920    text$(54)=' "SB" : Escape to Super Basic fo
r a while. You return  to  the  program    with CO
NTINUE. Z and M registers are unchanged. In SB you
 may  calcu-   late eg. "PRINT 34+45".'
7930    text$(55)=' "STO" : Store Z1 in Memory (1-9
). You are asked which memory . You  may   add a n
ote against the number to help you remember what i
s saved.'
7940    text$(56)=' "SUM" : Add Z1 to Memory (1-9).
 You are asked which Memory you want  to   work on
.'
7950    text$(57)=' "RCL" : Recall the value from t
he Memory location you chose and put  it   in Z1 f
or use in  subsequent calculation.'
7960    text$(58)=' "CMs" : Clear chosen Memory or
all Memories. You are asked which.'
7970    text$(59)=' "HELP" : Gives HELP on the Comm
and/Function you type after  "HELP".  If  no Comma
nd/Function is entered, "HELP" will give help on
calculations   in general.'
7980    END DEFine
7990 :
8000 DEFine PROCedure write(tekst$)
8010    CLS#6 : AT#6,0,0 : PRINT#6,tekst$
8020    AT#6,2,59 : PRINT#6,' Press a key '
8030    a$=INKEY$(-1) : CLS#6
8040    END DEFine
```

THE PROGS

If you have a program worthy of consideration, send it to 'The Progs', Sinclair QL World, Greencoat House, Francis Street, London SW1P 1DG. We pay for everything published at the usual page rates.

PLANT LIFE

by Colin Bates

Plant Life was one of the popular programs submitted to the *QL World* Artist of the Year 1988 competition. Although not one of the two prizewinners it made a good impression on everyone who saw it.

It is short and simple enough to type-in without tears and will keep your QL occupied attractively in its quiet moments.

When the program is run, tendrils of variegated foliage start to build from the bottom of the screen. When 'enough' foliage has developed, the plant produces exotic crimson blooms with translucent halos. Run the program again and a different version of the shrub appears. It is not fast and is ideal watching for people who enjoy the company of goldfish.

Plant Life is probably at its best on a wall-sized projection screen but an ordinary colour monitor will serve.

```
100 REMark     ***************************************
110 REMark     *****                             *****
120 REMark     *****  ... PLANT .... LIFE ...    *****
130 REMark     *****                             *****
140 REMark     *****      by Colin Bate          *****
150 REMark     *****                             *****
160 REMark     ***************************************
170 WINDOW #0,452,32,30,212
180 WINDOW #1,512,256,0,0
190 WINDOW #2,452,202,30,10
200 OPEN#3,SCR_512X256A0X0
210 FOR n=0 TO 1:  INK#n*2,7:  PAPER#n*2,0
220 MODE 4:  BORDER 30,7,0:  BORDER 29,0:  BORDER 27,2,
4,0:  BORDER 22,0:  BORDER 30
230 SCALE 196,0,0:  k=4:  k1=7
240 SCALE#3,256,-45,-30:  k=4:  k1=7
250 REMark
260 REMark          *****  PLANTS  *****
270 REMark
280 FOR tim=0 TO 7
290 q=RND*2*PI:  q1=RND*PI:  x=32+32*tim+RND(10)-RND(10
):  y=RND(10):  l=2+RND(20):  t=1-2*RND(1):  led=5+RND(
15):  SX=0:  BY=0
300 IF k1=4:  k1=5:  GO TO 320
310 k1=k1+2:  IF k1>5:  k1=0
320 led=6+RND(13-k1)
330 m1=15+RND(25):  df=(led-1)/m1:  s=1:  FOR m=1 TO m1
340 SX=SX+3.6:  BY=BY+4.8:  led=led-df:  IF led<7:  s=3
350 FOR n=q TO q+q1*t STEP 3*t/l
360 c=COS(n):  d=SIN(n):  c1=x+c*l:  d1=y+d*l:  c2=x+c*
(1+t*led):  d2=y+d*(1+t*led)
```

```
370 c3=x+c*(1-t):  d3=y+d*(1-t):  c4=x+c*(1+t*(led+t)):
  d4=y+d*(1+t*(led+t))
380 IF n<>q:  INK#s,0:  FILL#s,1:  LINE#s,c3,d3 TO c4,d
4 TO f4,g4 TO f3,g3 TO c3,d3:  FILL#s,0:  INK#s,k,k1:
FILL#s,1:  LINE#s,f1,g1 TO c1,d1 TO c2,d2 TO f2,g2 TO f
1,g1 TO f2,g2:  FILL#s,0
390 f1=c1:  f2=c2:  g1=d1:  g2=d2:  f3=c3:  f4=c4:  g3=
d3:  g4=d4
400 IF RND>.9:  leaf2
410 NEXT n
420 t=-t:  g=0:  gig=0
430 l1=l+RND(20+g)+2:  x1=x+COS(n)*l1:  y1=y+SIN(n)*l1
440 q=dif(n):  q1=.175+RND*PI:  IF RND>.9:  q1=.175+RND
*PI*3/2
450 x2=x1+COS(q+q1*t)*(l1-l):  y2=y1+SIN(q+q1*t)*(l1-l)
460 IF (x2<SX OR x2>290-SX) OR (y2<BY OR y2>195)
470 g=g+2:  IF g=20 AND gig=0:  g=0:  gig=1:  t=-t:  GO
 TO 430
480 IF g=20 AND gig=1:  SX=SX-3.6:  BY=BY-4.8:  GO TO 4
20
490  GO TO 430
500 END IF
510 x=x1:  y=y1:  l=l1-l
520 NEXT m
530 NEXT tim
540 REMark
550 REMark     *****  LIFES  *****
560 REMark
570 k=2:  k1=2
580 FOR tim=0 TO 7
590 radius=10+RND(23):  cx=-10+radius+RND(300-radius):
 cy=-10+radius+RND(200-radius):  q=RND*2*PI:  q1=RND*PI
:  l=2+RND*radius/2:  t=1:  led=1+radius/7:  FILL#3,0
600 INK#3,0:  ra=radius+radius/2+1:  FOR y=-ra TO ra ST
EP 1.5
610 x=(ra*ra-y*y)^.5:  LINE#3,cx-x,cy+y TO cx+x,cy+y
620 NEXT y:  y=cy:  x=cx
630 INK#3,7:  FOR n=ra TO ra-ra/60 STEP -ra/120:  CIRCL
E#3,cx,cy,n
640 x=cx:  y=cy:  ra=ra-2
650 FOR m=1 TO 1.75*radius
660 le=0
670 FOR n=q TO q+q1*t STEP 3*t/l
680 c=COS(n):  d=SIN(n):  c1=x+c*l:  d1=y+d*l:  c2=x+c*
(1+t*led):  d2=y+d*(1+t*led)
690 c3=x+c*(1-t):  d3=y+d*(1-t):  c4=x+c*(1+t*(led+t)):
  d4=y+d*(1+t*(led+t))
700 IF n<>q:  INK#3,0:  FILL#3,1:  LINE#3,c3,d3 TO c4,d
4 TO f4,g4 TO f3,g3 TO c3,d3:  FILL#3,0:  INK#3,k,k1:
FILL#3,1:  LINE#3,f1,g1 TO c1,d1 TO c2,d2 TO f2,g2 TO f
1,g1 TO f2,g2:  FILL#3,0
710 f1=c1:  f2=c2:  g1=d1:  g2=d2:  f3=c3:  f4=c4:  g3=
d3:  g4=d4
720 NEXT n
```

```
730 t=-t:  g=0:  gig=0
740 l1=l+RND*(radius/2-g)+2: .x1=x+COS(n)*l1:  y1=y+SIN
(n)*l1
750 q=dif(n):  q1=.195+RND*PI*3/2:  IF RND>.9:  q1=.175
+RND*PI
760 x2=x1+COS(q+q1*t)*(l1-l):  y2=y1+SIN(q+q1*t)*(l1-l)
770 rd=((x2-cx)*(x2-cx)+(y2-cy)*(y2-cy))^.5
780 IF rd+(l1-l)+ra/20>ra:  x=cx:  y=cy:  o=0:  ot=0:
q=RND*2*PI:  q1=RND*PI:  l=2+RND*radius/2:  GO TO 800
790 x=x1:  y=y1:  l=l1-l
800 NEXT m
810 dot
820 NEXT tim:  K$=INKEY$(-1):  STOP
830 REMark
840 REMark     ***** FUNCTIONS & PROCEDURES *****
850 REMark
860 DEFine FuNction dif(z)
870 IF z<0:  z=z+2*PI:  GO TO 870
880 IF z>2*PI:  z=z-2*PI:  GO TO 880
890 IF z<=PI:  RETurn z+PI*t
900 IF z>PI:  RETurn z-PI*t
910 END DEFine

920 DEFine PROCedure dot
930 FILL#3,0:  INK#3,7:  rd=100:  rd1=.25:  d=(rd-rd1)/
ra:  st=0
940 FOR n=1 TO ra
950 st=rd/n:  rd=rd-d:  IF st>2*PI:  NEXT n:  GO TO 102
0
960 an1=RND*2*PI
970 IF n<ra/2:  NEXT n:  GO TO 1020
980 FOR m=an1 TO 2*PI+an1-st STEP st
990 px=n*SIN(m):  py=n*COS(m)
1000 POINT#3,cx+px,cy+py
1010 NEXT m:  NEXT n
1020 FOR n=0 TO ra*8:  ra1=ra-ra/10-RND*(ra/5):  ang=PI
/10+RND*(PI/2-PI/20):  POINT#3,cx+ra1*SIN(ang),cy+ra1*C
OS(ang)
1030 INK#3,4:  FOR n=0 TO ra*8:  ra1=ra-ra/10-RND*(ra/5
):  ang=PI+PI/5+RND*(PI/2-PI/5):  POINT#3,cx+ra1*SIN(an
g),cy+ra1*COS(ang)
1040 END DEFine

1050 DEFine PROCedure leaf2
1060 c=COS(n):  d=SIN(n):  ca1=x+c*l:  da1=y+d*l:  le=1
1070 ca3=x+c*(l+t):  da3=y+d*(l+t):  ca4=x+c*(l-t*(led-
t)):  da4=y+d*(l-t*(led-t))
1080 c=COS(n-PI*t/4):  d=SIN(n-PI*t/4):  cb1=x+c*led*4:
 db1=y+d*led*4:  cb2=x+c*(led*4+3):  db2=y+d*(led*4+3)
1090 cb3=x+c*(l+t):  db3=y+d*(l+t):  cb4=x+c*(l-t*(led-
t)):  db4=y+d*(l-t*(led-t))
1100 FILL#3,1:  INK#3,0:  ARC#3,ca1,da1 TO cb1,db1,PI/1
.4:  ARC#3,ca1,da1 TO cb2,db2,-PI/1.4:  FILL#3,0
1110 FILL#3,1:  INK#3,k,k1:  ARC#3,ca1,da1 TO cb1,db1,P
I/2:  ARC#3,ca1,da1 TO cb1,db1,-PI/2:  FILL#3,0
1120 END DEFine
```

TURBOQUILL

David Cottom investigates a low-cost solution to the QL user's favourite complaint – Quill speed.

When the QL was launched, it broke new ground in both packaging and pricing, with its inclusion of the Psion suite of word processing, spread-sheet, database and business graphics in the price of £399. Even today when there are faster, industry-standard PCs and more comprehensive and integrated packages on the market, the QL still stands out as excellent value, with new QLs available for around £100 including the Psion four principal applications.

Surveys indicate that most personal computer users spend more than half their time at the keyboard word processing, so it is probably not surprising that the Quill component of the Psion suite has received most criticism from QL users. The principal complaint is its poor performance – its sheer lack of speed.

Anyone who has watched Quill in action – as opposed to just having used it – will have some idea as to the reason. Quill writes to the screen constantly and not just the text being typed – the word, line and page counts, command and typeface prompts and the optional, boxed command area at the top of the screen are being updated permanently.

Speed

Experienced users will already have gained a significant increase in speed, particularly when using a number of commands, by removing the boxed prompts with the F2 option, but Quill is still sluggish. Try setting up a text string to be typed-in automatically by an ALTKEY function, such as that in *Toolkit II* and you will see what I mean. *SpeedScreen* and *Lightning* offer some improvement but the fundamental problem is the internal inefficiency of Quill.

There are two approaches for users who can live with that poor performance no longer. The first is to buy an alternative, one of the many editors available for the QL, most of which have varying degrees of word processing capability, or the highly-regarded *text 87*.

That can be an expensive option, with most editors costing around £30 and the most expensive priced at almost double that. Of course, editors have other uses for which Quill is not suitable, for example to input and edit program source listings, particularly for languages other than SuperBasic, and to edit binary program files.

For the user on a budget, who wants to do occasional word processing, including longer items such as reports, using the software with which he is familiar but with improved performance there is a second, cheaper option.

TurboQuill, for users with standard QLs, and its enhanced derivative, *TurboQuill+*, for users with expanded memory, work by patching the standard Quill code with fast compiled Pascal routines which speed, by replacement, some of the most inefficient and frustrating Quill processes. In addition, TurboQuill+ adds some extra features such as a CAPS LOCK indicator and an easy-to-use Glossary – key define – function.

As a *TrumpCard* user, I ordered the latter version which was supplied by PDQL for £13. Note that TurboQuill and TurboQuill+ are Quill version-dependent, so be sure to order either version 2.3 or 2.35 as appropriate. In my case, as version 2.3 of TurboQuill+ was not in stock, PDQL supplied version 2.35 of Quill on the same disc.

TurboQuill+ is supplied on disc or Microdrive with the instructions saved in a Quill document. Installation is a question and answer process, which is probably best performed from boot as the Pascal run-time library must be loaded and linked in before use.

Multi-tasking

Boot the TurboQuill+ disc or Microdrive and answer the questions; device and filename of the Quill to be modified, whether or not you want full CAPS LOCK functionality – the instructions warn that programs such as key defines, *Ice* and most multi-tasking systems other than *TaskMaster* may have problems if this option is chosen – and whether the modified Quill is to be run under a QRAM environment. The specifield Quill, which must be the correct version for your TurboQuill+, is then modified and saved to the device and filename entered.

Full CAPS LOCK functionality means that the normally red text cursor will change colour to white immediately the CAPS LOCK key is pressed. If you choose not to implement full CAPS LOCK functionality, as I was compelled to do as a QRAM user, the cursor changes colour only after a character key has been pressed. This delayed change detracts little from this useful, almost incidental, facility.

QRAM users should note that a version of TurboQuill+ created with the QRAM option set will work only under QRAM, so if you also use TurboQuill+ without QRAM present , you will need to create a second version without this option set.

Once your Quill has been converted, the standard Psion config_bas program will not work, so TurboQuill+ is supplied with its own configuration program, this time in Pascal. This program allows an additional option to be set, the location of the Glossary file, the default being FLP1_, although I chose to use RAM disc for maximum speed.

As usual, QRAM users must configure TurboQuill+ before using Grabber to allocate the amount of memory to be used by the program. It is important to note that TurboQuill+ is slightly more than 8K larger than standard Quill, so you must specify that amount more memory to retain the same maximum size for a fully memory-resident document.

A modification program is supplied for *SpellBound* users which enables SpellBound to be used with a TurboQuill+ which has full CAPS LOCK functionality set. The number and comprehensive nature of these various configuration options is an indication of the thoroughness with which TurboQuill+ has been prepared.

Faster

First impressions of TurboQuill+ are that it is significantly faster than standard Quill. Most noticeably it is now possible to work with the boxed command area at the top of the screen present, something I still find useful even after using Quill for four years, at home and at work. The faster prompt display is not the only improvement as use of ALTKEY functions shows; speed of text input to the document is also improved significantly.

To measure the performance improvement, I set up and ran several of these functions in both standard Quill and TurboQuill+, with and without the boxed prompts present, and with and without *Lightning* installed. Each test was run three times in a 896KB QL with no other tasks running to ensure the validity of the timings. The results of a typical function to format a letter, resetting the margins, removing the footer, changing the design and typing-in my name, address and telephone num-

	Without Lightning		With Lightning	
	With Prompts	No Prompts	With Prompts	No Prompts
Quill	22 sec.	13 sec.	18 sec.	11 sec.
TurboQuill+	10 sec.	7 sec.	10 sec.	7 sec.

Comparative timings for a typical ALTKEY function.

ber, are summarised in the Table.

The surprising aspect of these results is not the improvement in speed offered by TurboQuill+ – around twice as fast as standard Quill on average – but the fact that once Turboquill+ has been installed, Lightning provides no measurable additional increase in speed, at least for functions of this length. It is possible that a longer benchmark would have produced a measurable additional speeding but few people would use ALTKEY functions of that length and no typist could input keystrokes faster than an ALTKEY function.

In addition to the CAPS LOCK indicator, TurboQuill+ also provides a Glossary feature which allows up to 22 functions to be defined to perform frequent or repetitive actions automatically, such as the ALTKEY functions described in the performance measurements.

Many users will already have access to the ALTKEY function which is part of Toolkit II, fitted as standard on many disc interfaces, and is also provided by various Key Define programs. Unlike most of these, the TurboQuill+ Glossary function uses the CTRL key rather than ALT and the definitions are case-independent.

Only letters from A to Z may have functions defined, excluding CTRL/C for obvious reasons and CTRL/I, CTRL/J and CTRL/O. Users who find the resulting 22 functions limiting may still use ALTKEY function as well, offering almost unlimited options. The function defined by CTRL/S is executed automatically on start-up, providing a way to eliminate some of the annoying Quill defaults.

The most noticeable difference between ALTKEY key definition and the TurboQuill+ Glossary feature is the ease with which functions are defined. Whereas setting up ALTKEY definitions can be tedious, in the case of Toolkit II having to be typed into Super-Basic, usually as part of the BOOT program, and requiring non-character keys to be entered as CHR$(key-code), the Glossary is easy by comparison.

It works rather like a tape recorder. Press F5 to open the Glossary window – see figure one – press the letter for which you wish to define the function, then the space bar to turn on recording, and finally ESC to close the window. All key depressions are now stored in the Glossary file. To finish recording, press F5 then ESC.

Key definitions are stored on the device specified in the TurboQuill+ configuration program. If, like me, you choose RAM disc for maximum speed you must remember to copy the Glossary file to the specified RAM disc before using it – for example by including it in the BOOT file – and to copy it back to permament storage before switching off if you have made changes.

Locked up

After several months of fairly heavy use TurboQuill+ has never once locked up on me or done anything unexpected, with one exception. HELP screens, called up with the F1 key, are all underlined word by word. This is scarcely a great inconvenience to experienced users because after the first two hours you rarely use HELP again. I uncovered this 'feature' only when my wife was learning to use Quill. Other than this, TurboQuill+ is, in my experience, bug-free.

Slicker

TurboQuill+ is a remarkable program which transforms Quill from an almost intolerably slow word processor into a much slicker application. It may not have all the speed and functionality of some of its competitors on the QL but its advantages are that it looks and works exactly the same, so there is no learning cycle to go through, and its low price. Some useful extra functionality is there for good measure. I wonder how I managed with the original for so long.

INFORMATION:
Program: TurboQuill+.
Supplier: PDQL, Unit 1, Heaton House, Camden St., Birmingham B1 3BZ. Tel. 021 200 2313.
Price: £13.

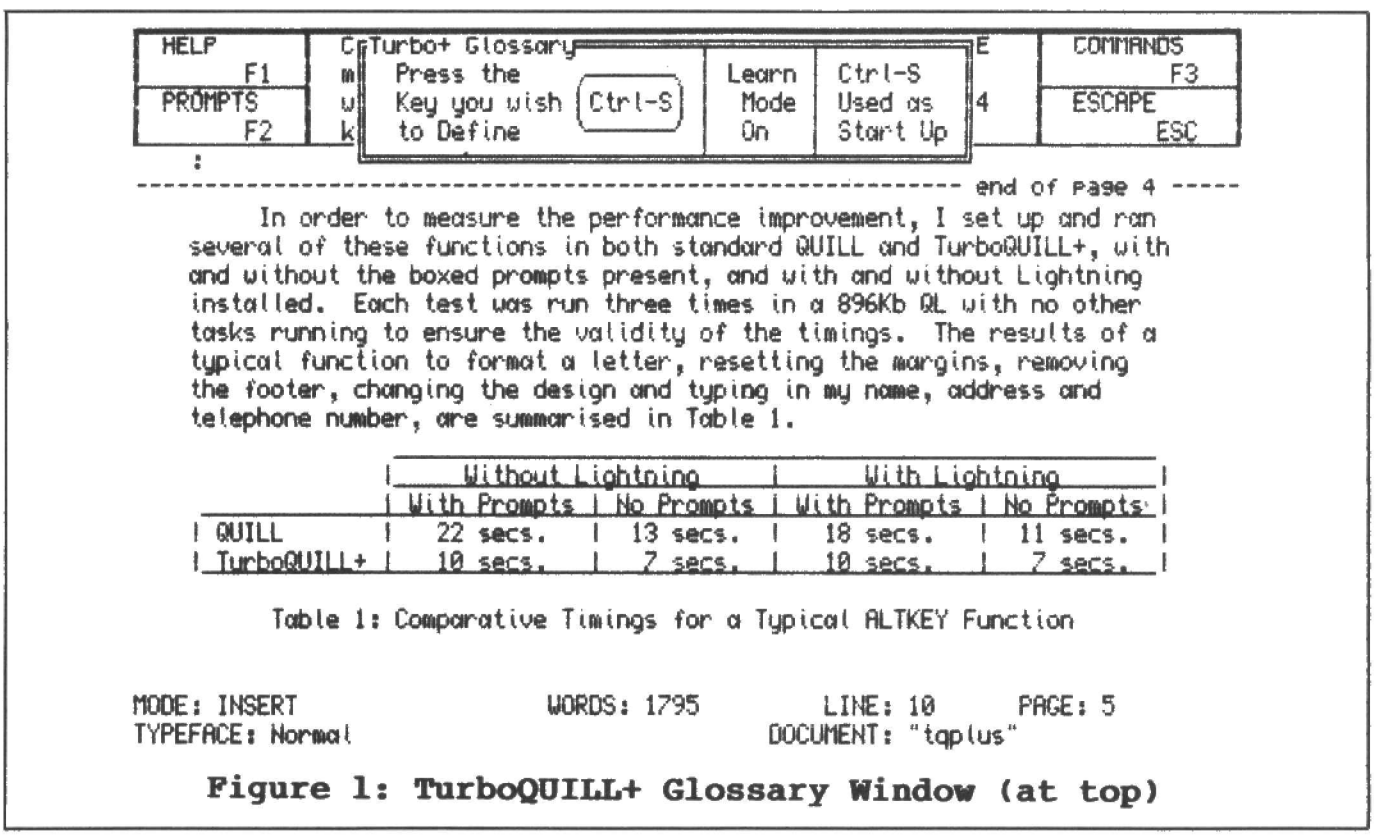


Figure 1: TurboQUILL+ Glossary Window (at top)

PDQL Computer Systems and Software

UNIT 1, HEATON HOUSE CAMDEN STREET BIRMINGHAM B1 3BZ 021 200 2313

PDQuaLity Products

PDQL Titles

Appointments Diary £50
Archive Database Analyser £10
Archive Screen Format Printer £50
Archive Tutorial £21
Cash Trader Upgrade £85
CT Analyser £25

Special discounts for Quest users

DiscOVER £29.50
Domination £9
FileBound (your disc/cartridge] £5
graFix £20
Hardback and Finder £35
HEX Dump to Printer £10
Home Budget £20
Lazarus £20
Mailmerge de Luxe £20
MULTI-DiscOVER £39
Names and Addresses £20
With Archdev RTM extension £36
PDQ-C £79
PDQ-Copy £10
PDQ-Payroll £80
Psion Printer Installer/Configurator £6
Recover £20
SEDIT £20
SuperBASIC C-PORT £79
Toolkit library modules
Turbo and TK2 – each £30
SuperBASIC Monitor £10
TexTIDY £10
Trading Accounts £125
Xref 200 ..... £20
Xref 300 ..... £25

HARDWARE

Miracle

40MB Hard Disc £450
Trump Card – 768K £249
Trump Card – 512K £199
Trump Card – 256K £129
Trump Card – 768K with twin 3.5in. disc drives. £425
Trump Card – 512K with twin 3.5in. disc drives. £349
Trump Card – 256K with twin 3.5in. disc drives. £299
Zero Trump Card – Discard £100
Twin 3.5in. disc drives £181

Other recommended software titles

QTYPE £29
Spellbound £30
Flashback Old £25
Flashback New £40
Page Designer 2
Clip Art each £6
Image Processor £19
Taskmaster £25
text 87 ..... £45
fountext 88 ..... £25
founted 89 ..... £15
The above three £80
typeset 2488 ..... £15
The Editor £45
The Conqueror – Vanilla £89
– Chocolate £139
TURBO £99
Pro Publisher £89
3-D Precision £49
Lightning – standard £29
– special £49
Eye-Q £29
TurboQuill+ £13
Cartridge Doctor £13
ArchDEV/rtm £20
Archive v.2.38, Quill/Easel/Abacus v v.2.35 (on your cartridge/disc) ..... £5 each
King James Bible and Index £45

Other titles on request

Tailor-made systems a speciality

Monitors – prices on application

STAR printers mono, colour, 9 pin, 24 pin
At competitive prices

Introducing the **NEW LC-10 II** £265

Archive and Utility Packages – Details inside QL World

Write or phone with order – payment by ACCESS or VISA – cheque/Eurocheque
Prices incl VAT & UK carriage/postage – All orders subject to PDQL's Sale Conditions